Aalto University
School of Science
Master's Programme in Engineering Physics

Niko Oinonen

Interpreting atomic force microscope images with machine learning

Master's Thesis
Espoo, 08.08.2019

Supervisor: Professor Adam Foster
Advisors: Ph.D Prokop Hapala
Ph.D Fedor Urtev

Aalto University
School of Science
Master's Programme in Engineering Physics

ABSTRACT OF
MASTER'S THESIS

Author: Niko Oinonen

Title:
Interpreting atomic force microscope images with machine learning

Date: 08.08.2019 **Pages:** 42

Major: Engineering Physics **Code:** SCI3056

Supervisor: Professor Adam Foster

Advisors: Ph.D Prokop Hapala
Ph.D Fedor Urtev

Since its invention in 1986, atomic force microscopy (AFM) has developed into a unique tool for exploring the microscopic world. With the introduction of CO tip functionalization, the image resolution has reached the level of individual atoms and bonds. However, the use of this method so far has been mostly restricted to planar structures, due to difficulties in interpretation of images for more complex 3D molecular structures.

We aim to address this problem with the use of artificial neural networks (ANN), a type of machine learning model. ANNs have gained much attention in recent years for advancing the state of the art in many complex problems, including those related to natural language processing, image recognition, autonomous cars, and playing games at a superhuman level. The success of ANNs has been enabled by the increased availability of computational resources and datasets of sufficient size.

In the work of this thesis, we apply convolutional neural networks, a type of ANN, to the task of predicting easily interpretable descriptors of atomic properties from AFM images. The models are trained on simulated AFM images and tested on both simulated and experimental images. The results on simulated images are generally very good, but experimental results, while in some cases promising, indicate that there are some challenges that need to be overcome.

Keywords: scanning probe microscopy, atomic force microscope, machine learning, deep learning, convolutional neural network

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Engineering Physic

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Niko Oinonen		
Työn nimi:	Atomivoimamikroskooppikuvien tulkitseminen koneoppimisen avulla		
Päiväys:	08.08.2019	**Sivumäärä:**	42
Pääaine:	Engineering Physics	**Koodi:**	SCI3056
Valvoja:	Professori Adam Foster		
Ohjaajat:	Tohtori Prokop Hapala Tohtori Fedor Urtev		

Atomivoimamikroskopiasta (eng. atomic force microscopy, AFM) on muodostunut tärkeä menetelmä mikroskooppisen maailman tutkimiseen. Jos AFM-laitteen neulankärkeen kiinnitetään joustava hiukkanen, kuten CO-molekyyli, voidaan saavuttaa tarkkuus, joka yltää yksittäisten atomien ja sidosten tasolle. Menetelmän käyttö tähän asti on kuitenkin rajoittunut vain verrattain tasaisten rakenteiden kuvaamiseen, koska monimutkaisempien rakenteiden AFM-kuvat ovat yleisesti vaikeasti tulkittavia.

Tässä diplomityössä etsitään ratkaisua edeltävään ongelmaan käyttämällä neuroverkkoja. Neuroverkot ovat koneoppimismalleja, jotka ovat nopeasti edistäneet tutkimusta monissa tekoälyyn liittyvissä ongelmissa, mukaan lukien tekstinymmärryksessä, tietokonenäössä, itseohjautuvien autojen kehityksessä ja pelien pelaamissa. Kehityksen ovat mahdollistaneet erityisesti kasvaneet ainestomäärät ja grafiikkapiirien mahdollistama laskentakapasiteetin kasvu.

Tässä työssä opetetaan konvoluutioneuroverkkomalleja tunnistamaan AFM-kuvista helposti tulkittavia kuvauksia atomirakenteista. Mallien oppiminen tapahtuu simuloitujen AFM-kuvien avulla, ja malleja testataan sekä simuloiduilla että kokeellisilla kuvilla. Tulokset ovat yleisesti hyviä simuloiduilla kuvilla, mutta kokeelliset kuvat osoittautuvat vielä usein haastaviksi.

Asiasanat:	atomivoimamikroskooppi, koneoppiminen, syväoppiminen, konvoluutioneuroverkko
Kieli:	Englanti

Acknowledgements

The work leading up to this thesis was done in the Surfaces and Interfaces at Nanoscale (SIN) group of the Department of Applied Physics in Aalto University, lead by Professor Adam Foster. None of this would have been possible without the contributions from a number of people. Here is just a brief word of thanks for all the help I have got. Thanks to Ondřej Krejčí for getting me initially started on the simulations and giving me feedback on the writing of this thesis. Thanks to Filippo Federici Canova for providing the molecule structures used in the simulations. Thanks to Benjamin Alldritt and Peter Liljeroth from the Atomic Scale Physics group for providing the experimental data. Thanks to Juho Kannala from the Department of Computer Science for his expertise in machine learning. A special thanks to the advisors of this thesis, Prokop Hapala and Fedor Urtev, for all their guidance and their essential contributions to the project. And of course, thanks to Adam for creating the opportunity for all these people to come together. Finally, I acknowledge the use of the computational resources in the Aalto University School of Science Science-IT project.

Contents

1 Introduction

Atomic force microscopy (AFM) has developed into an important tool for characterising microscopic structures on surfaces [1]. In AFM, an atomically sharp tip attached to a cantilever is brought very close to a sample such that the force interaction between the atoms in the tip and the sample produces a measurable signal [2]. In this thesis we focus on frequency modulation AFM [3], a technique that has reached reliable atomic resolution in imaging individual molecules. This has been enabled by the introduction of tip functionalization, where a chemically inert and flexible tip apex, such as a CO-molecule, is attached to the tip [4]. In addition to structure discovery, high-resolution AFM has found use in studying, for example, chemical properties such as bond order [5], on-surface chemical reactions [6], and biological materials [7].

In the simplest cases of mostly planar molecules, the signal can be interpreted straightforwardly to learn the atomic structure [4]. However, with more complicated structures, the measured signal becomes significantly more difficult to translate to an understanding of what kind of molecular structure is in fact being imaged [8, 9]. The goal of this thesis is to tackle this inverse problem with the use of machine learning, and more specifically, artificial neural networks (ANNs). With this approach, AFM with flexible tip apexes could work as a solitary or complementary method to understanding 3D molecular structures.

In recent years, ANNs have significantly advanced the state of the art in a variety of highly complicated tasks [10]. In particular, convolutional neural networks (CNNs), a subset of ANNs, have been successful in solving problems of computer vision [11–16]. The power of ANNs comes from their ability to learn a highly non-linear, end-to-end hierarchy of representations of data without the need of feature engineering. The success of ANNs in recent years has been in large part due to increased availability of large datasets and the utilization of the parallel-computing capability of graphics processing units (GPUs) in the training of the models.

We design several descriptors that characterise the atomic positions and charge distribution of molecules in an easily interpretable way, and train CNN models that can predict these descriptors from sets of constant-height AFM images. The training data is generated with simulations [17, 18] from a large database of 3D molecular structures. The models are validated on a test set of simulated images and experimental images of 1S-camphor deposited in several configurations on a Cu(111) surface. The results are promising, but there are some challenges to overcome.

The rest of this thesis is structured as follows. Section 2 explains the background for the methods used in this thesis. Section 3 describes the more detailed methods related to the simulation model, the descriptors, the machine learning models, and how we process the data. Section 4 describes the results for each machine learning model. Finally, Sec. 5 provides some discussion on the results and some possible future directions in improving our methods.

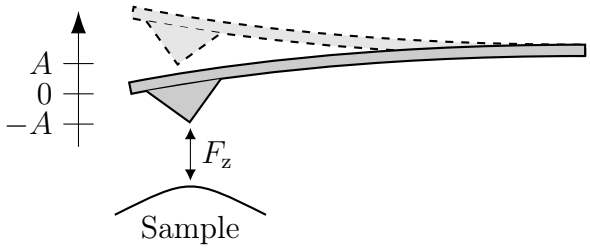

Figure 1: Schematic of an AFM device with a cantilever oscillating above a sample.

2 Background

The topic of this thesis is at the cross-section of two fields of study: atomic force microscopy and machine learning. This section provides the theoretical background for understanding these two domains.

2.1 Atomic force microscopy

Compared to methods like optical microscopy and electron scanning microscopy, which work by "seeing" the sample by detecting particles scattering off the sample, the principle of an AFM works more like "touching". An AFM device consist of an atomically sharp tip attached to a cantilever that can be moved both laterally and vertically over the sample. When the tip is brought very close to the sample, the interaction force between the atoms in the tip and the sample causes measurable change in the vertical position or motion of the cantilever. The idea is illustrated schematically in Fig. 1. An advantage of AFM over methods that require irradiation is that the spatial resolution is not limited by the diffraction limit and it is possible to image samples that could be damaged by the irradiation.

The AFM is a member of a larger class of microscopy techniques known as scanning probe microscopy (SPM). The first SPM device realized was the scanning tunnelling microscope (STM), invented in 1981 by Binnig et al. [19]. The inventors were awarded the Nobel prize in physics for their work in 1986, the year which also saw the introduction of the AFM [2]. An STM has a device structure similar to an AFM, but instead of the force, the measured signal is the tunnelling current between the tip and the sample. The relationship between the distance and the tunnelling current in the simplest cases is exponential, a monotonic function. In comparison, the force interaction measured in AFM is a non-monotonic function of the distance, arising from many origins, including van der Waals forces, electrostatic forces, Pauli repulsion, and chemical interactions. The use of an STM is limited to materials which are electrically conductive, while no such limitation exists for the AFM.

Expressed in terms of the interaction potential V and the tip-sample distance z, the vertical component of the tip-sample force is $F_z = -\partial V/\partial z$. In molecular dynamics simulations [20] the contribution from the van der Waals force and Pauli repulsion to the potential are approximated with pair-wise interaction potentials

such as the Lennard-Jones potential

$$V_{\mathrm{LJ}}(r) = \varepsilon \left(\frac{r_{\mathrm{m}}^{12}}{r^{12}} - 2\frac{r_{\mathrm{m}}^{6}}{r^{6}} \right), \tag{1}$$

where r is the the distance between the pair of atoms and r_{m} and ε are parameters that determine the length and energy scale, respectively. The r^{-12} term approximates the Pauli repulsion, and the r^{-6} term describes the van der Waals force. The electrostatic potential can be approximated by Coulomb potential with point charges on all atoms. There also exist methods such as ReaxFF [21], which additionally model chemical interactions between atoms. Another alternative that encompasses all of the interactions is doing *ab initio* quantum mechanical calculations [20], but this is computationally very expensive. The force interaction can also be understood via Hooke's law with an effective spring constant $k_{\mathrm{ts}} = -\partial F_{\mathrm{z}}/\partial z$. [3]

In static AFM, the surface of constant deflection above the sample is recorded, interpreting this as an isosurface of the tip-sample interaction F_z through Hooke's law. In order to prevent deformations of the tip and the sample, the spring constant of the cantilever k_{c} should be smaller than the force constants of the bonds in the tip and the sample. On the other hand, if the cantilever is too soft, $k_{\mathrm{c}} < \max\{k_{\mathrm{ts}}\}$, the tip can suddenly jump to contact with the sample. This can be avoided with a dynamic mode of AFM, where the cantilever is vertically oscillated, either in amplitude modulation (AM) or in frequency modulation (FM) mode. In this case the spring constant of the cantilever is also typically much higher. [3]

In AM-AFM, the cantilever with eigenfrequency f_0 is driven with a signal of constant amplitude and frequency close to but not the same as f_0. The tip-sample interaction causes a change in the amplitude and phase of the cantilever oscillation, which are the measured signals. By contrast, in FM-AFM the cantilever is driven at the resonance frequency with positive feedback such that the amplitude stays constant. The measured signal is the frequency shift $\Delta f = f - f_0$ of the cantilever from its eigenfrequency. AM-AFM has an inherent disadvantage in that the time scale of the changing of the amplitude is very slow. This is especially true in ultra-high vacuum conditions, where FM-AFM is the preferred technique. [3]

The combined system with the oscillating cantilever and the tip-sample interaction form a harmonic oscillator with an effective spring constant k^*, an effective mass m^*, and an eigenfrequency given by

$$f = \frac{1}{2\pi}\sqrt{\frac{k^*}{m^*}}. \tag{2}$$

If k_{ts} is constant throughout the oscillation cycle, then $k^* = k_{\mathrm{c}} + k_{\mathrm{ts}}$. If we further approximate that $k_{\mathrm{c}} \gg k_{\mathrm{ts}}$ and $m^* \approx m$, where m is the mass of the cantilever, then

$$\begin{aligned} f &= \frac{1}{2\pi}\sqrt{\frac{k_{\mathrm{c}}}{m^*}}\sqrt{1 + \frac{k_{\mathrm{ts}}}{k_{\mathrm{c}}}} \\ &\approx \frac{1}{2\pi}\sqrt{\frac{k_{\mathrm{c}}}{m}}\left(1 + \frac{k_{\mathrm{ts}}}{2k_{\mathrm{c}}}\right) \\ &= f_0\left(1 + \frac{k_{\mathrm{ts}}}{2k_{\mathrm{c}}}\right), \end{aligned} \tag{3}$$

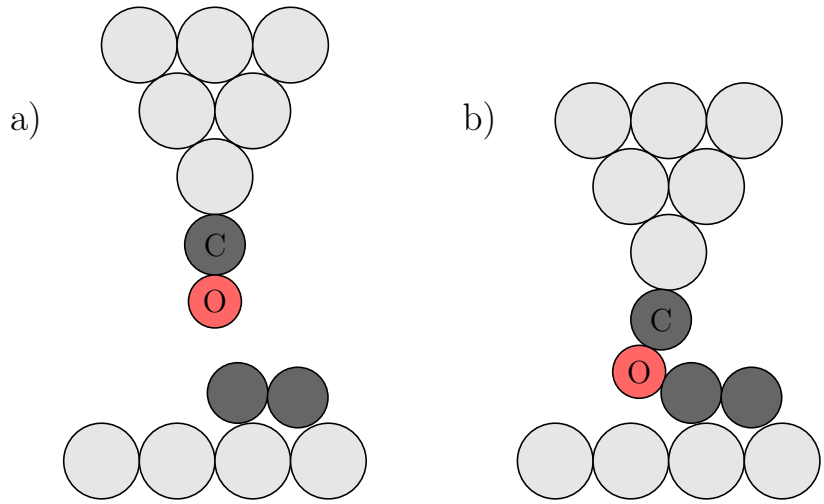

Figure 2: Illustration of the imaging mechanism of a functionalized AFM tip. a) At far distance, the interaction with the sample is weakly attractive. b) At close approach, the flexible tip apex bends around the sample atoms due to Pauli repulsion.

so that the frequency shift is

$$\Delta f = \frac{k_{\mathrm{ts}}}{2k_{\mathrm{c}}} f_0. \tag{4}$$

More generally, when k_{ts} is not constant, the frequency shift is given by Giessibl's formula [22]

$$\Delta f(z) = -\frac{f_0}{2k_{\mathrm{c}}} \frac{2}{\pi A^2} \int_{-A}^{A} \frac{q F_{\mathrm{z}}(z-q)}{\sqrt{A^2-q^2}} dq \tag{5}$$

$$= \frac{f_0}{2k_{\mathrm{c}}} \frac{2}{\pi A^2} \int_{-A}^{A} k_{\mathrm{ts}}(z-q)\sqrt{A^2-q^2} dq, \tag{6}$$

where A is the oscillation amplitude of the cantilever. The formula is of the same form as the simplified one in Eq. (4), with the difference that the force constant $k_{\mathrm{ts}}(z)$ is averaged over the oscillation path as a convolution with a weighting function.

It has been found that functionalizing the tip with a flexible and chemically inert tip apex is essential for achieving sharp contrast in high-resolution AFM. The chemical structure of an individual molecule on a surface was first imaged at atomic resolution with a CO-molecule-functionalized tip by Gross et al. in 2009 [4]. The flexibility of the tip apex allows it to bend under the repulsive force at close approach to the sample atoms. The result is that a strong force interaction is only observed directly above repulsive potential ridges, yielding a strong contrast between regions with and without atoms. Figure 2 illustrates this idea and Fig. 3 shows an example of the high contrast observed in AFM images. This mechanical-relaxation picture of the imaging mechanism is also supported by computational models [17].

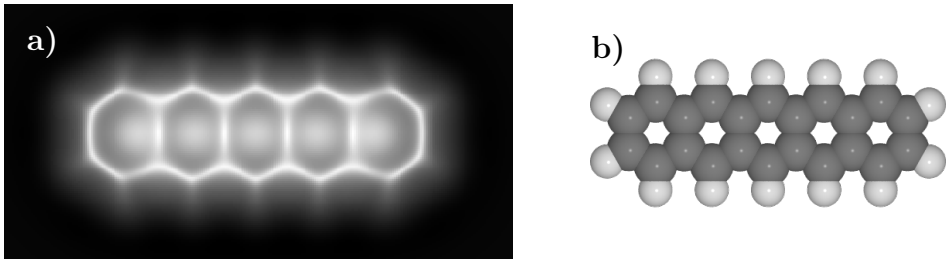

Figure 3: a) A simulated AFM image and b) the molecular geometry of the pentacene molecule. The AFM image has a sharp contrast over the atoms and bonds in the carbon ring structure.

2.2 Machine learning

Machine learning consists of a wide range of different methods, most of which will not be covered here. Instead, this section provides an overview of the general principles and motivations in machine learning and describes more in detail the theory of artificial neural networks and convolutional neural networks, which are used in the computational experiments in this thesis.

2.2.1 General principles

Traditional computer programming is based on defining a logical sequence of actions to manipulate data in a way that leads to a useful outcome. However, when the tasks and data grow more complex, it gets more and more difficult to hard-code such a set of actions. Approaches that have the capability of extracting useful patterns from data without hard-coded knowledge are referred to as machine learning. In some sense this kind of pattern extraction attempts to make the learning process more human-like, since we as humans typically learn new concepts and skills more intuitively based on examples rather strict definitions.

Mathematically formulated, the task of machine learning is to estimate a function $F : \mathcal{X} \to \mathcal{Y}$ from a set of examples $X \times Y \subset \mathcal{X} \times \mathcal{Y}$ that satisfy $F(x) = y,\ \forall (x, y) \in X \times Y$. In a typical example, one might want to classify photographs based on whether they contain a cat or a dog. In this example, $\mathcal{X}$ is the set of all photographs containing a cat or a dog, and $\mathcal{Y} = \{\text{cat}, \text{dog}\}$. Since the photographs have a finite resolution and color depth, the set of all possible functions is finite, but it is quickly seen that trying all the different possibilities is not feasible. In many applications the size of the domain or codomain may even be infinite.

In machine learning the approach for estimating the true function F is to provide some model function f with parameters tuned using a set of example inputs and possibly outputs. Perhaps the simplest model would be a linear function $f(x) = ax + b$, where the tunable parameters are a and b. If we are given a set of paired example inputs X and outputs Y, and some goodness-of-fit metric, in machine learning context referred to as the loss function, we should be able to find optimal

values for a and b. A typical loss function to use is the mean squared error

$$\mathcal{E}(X, Y; w) = \frac{1}{N} \sum_{i=1}^{N} \| f_w(x_i) - y_i \|_2^2, \tag{7}$$

where $(x_i, y_i) \in X \times Y$ are pairs of examples, $N = |X| = |Y|$, and w is a vector of parameters. The problem becomes one of minimizing the loss function.

In the above case of linear function and mean squared error, the problem boils down to a least-squares problem, which has an exact solution. However, most practically useful models, such as the neural networks discussed below, are highly non-linear. In this case, there is no exact solution, and iterative methods must be used instead. The most often used method is gradient descent and variants of it. The idea of gradient descent is to make some initial guess on the parameters w_0 and then to iteratively update the parameters by taking a step in the opposite direction of the gradient,

$$w_{i+1} = w_i - \alpha \nabla_w \mathcal{E}(X, Y; w). \tag{8}$$

Here, α is a hyperparameter controlling the size of the step, and is often referred to as the learning rate. Higher learning rates lead to faster convergence, but too high a learning rate leads to the parameters overstepping the minimum or even diverging. There exist several more advanced variants that may take the parameter values from past steps into account or adapt the learning rate automatically for each parameter [23].

The parameter update rule in Eq. (8) uses all the examples to perform the update step. However, often due to the complexity of the model or the large amount of data, it is not computationally efficient to use all the data for every step. Instead, one might use stochastic gradient descent where just one sample per step is used or minibatch gradient descent where some small subset of samples per step are used. One full pass over all the samples is referred to as an epoch. In these methods, the iteration process is noisier, but is overall much faster and does not get stuck in local minima as easily.

Even if the parameters have been fitted to the examples with low loss, it is not yet clear if the model can generalize to new data, because the parameters could be overfitted to the data used for training. To overcome this problem, the data is usually divided into training, validation, and test sets. The training set is only used for fitting the model parameters. The validation set is used for tuning hyperparameters, i.e., parameters of the model that are not learned from the data, and monitoring if the model is overfitting. The test set is the final measure of the performance of the model, and no parameters are fitted to the this data. Generally, overfitting is reduced if the size of the training set is grown, and the model can also be regularized, for example, by adding a penalty for large values of the parameters or adding random noise to the training examples.

There are several different classes of machine learning models which have been used in a variety of applications. Machine learning approaches can be roughly divided into two categories: unsupervised and supervised. Unsupervised models learn patterns and properties from the features provided in the training set without explicit targets. Supervised models additionally have some target or label which

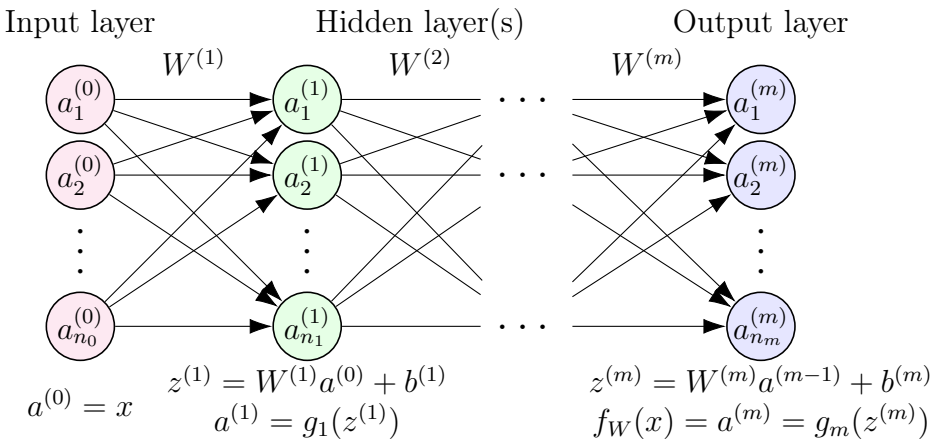

Figure 4: Illustration of the structure of an artificial neural network

they learn to predict based on the training set. Some of the numerous approaches include logistic regression, kernel regression, Bayesian models, support vector machines, clustering and artificial neural networks. We will discuss the last one in more depth in the following.

2.2.2 Artificial neural networks

One particular class of machine learning models is the artificial neural network (ANN). The name comes from the fact that their structure is inspired by the biological neural networks in the brain. ANNs have been very successful in performing a variety of tasks that are often associated with the study of artificial intelligence. Some of these tasks include image classification [11–13], machine translation [24, 25], autonomous cars [26] and playing games [27, 28]. The advantage of an ANN over many other methods is that it can learn a hierarchical representation of the data, from the data, instead of having to manually engineer features before the learning.

The idea of an ANN is not new. The first model of an artificial neuron was introduced by McCulloch and Pitts in 1943 [29]. Their model of a neuron was essentially a binary classifier on a weighted sum of binary inputs. In 1958 Frank Rosenblatt built a machine named perceptron that could learn the weights of a linear binary classifier [30]. The machine updated the weights with a rule that was inspired by Hebbian theory of learning, where the strength of connections between correlated neurons are increased. The modern software counterparts of the perceptron, often referred to as multi-layer perceptrons, consist of multiple non-linear functions and can be optimized with gradient-based methods [31]. ANNs have been gaining in attention in recent times due to increased availability of large datasets and increased computational capability, both critical factors for successfully training a complex network. In modern context, the domain of research and application of ANNs is referred to as deep learning, owing to the depth of the networks used [10].

An ANN divides the problem of function estimation into finding a composition of m functions, $f = f_1 \circ f_2 \circ \cdots \circ f_m$. Each function is represented by a layer in the network, as illustrated in Fig. 4. Each layer consists of several neurons, the

values of which are gathered in vectors $a^{(l)}$. The inputs in the first layer are the features x from the dataset and the values in the last layer are the output $f_W(x)$ of the network. All the layers between the input and the output are referred to as hidden layers. The connections between adjacent layers are parametrized by weight matrices $W^{(l)}$ such that each connection gets one weight. Each layer also has a bias vector $b^{(l)}$ with one parameter for each neuron. The values of the neurons in the next layer are calculated as

$$a^{(l)} = g_l(W^{(l)}a^{(l-1)} + b^{(l)}), \tag{9}$$

where $g_l(z)$ are some non-linear functions, referred to as activation functions. The output of an ANN is

$$f_W(x) = a^{(m)}(x) = g_m(W^{(m)}g_{m-1}(\dots g_1(W^{(1)}x + b^{(1)})\dots) + b^{(m)}). \tag{10}$$

Here the necessity of the activation functions is evident: without them the weight matrices could be multiplied together to produce just a single weight matrix, making the multiple weight matrices redundant.

There are several activation functions that are often used. A typical example would be the sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{11}$$

The sigmoid function takes values in range $(0, 1)$, which captures the intuition of neuron activity. A value close to 0 corresponds to an inactive neuron, and a value close to 1 corresponds to an active neuron. However, the sigmoid function and other similar functions have a disadvantage in that their gradient is very small away from the origin. This has resulted in the vanishing gradient problem in deep neural networks, where early layers train very slowly due to small gradients [32]. An alternative which mitigates this problem is the Rectified Linear Unit

$$\mathrm{ReLU}(z) = \max(0, z). \tag{12}$$

The ReLU function has the advantage that the gradient is constant so it does not slow down learning even in deep networks. However, the gradient is zero when $x < 0$. This can sometimes cause a part of the neurons in the network to "die" by always having an activation of zero and not being able to update. To solve this problem, it is possible to add a small contribution on the negative side, to allow for non-zero gradient even when the neuron is not active [33]. This is the Leaky Rectified Linear Unit function

$$\mathrm{LeakyReLU}_\alpha(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha z & \text{if } z < 0. \end{cases} \tag{13}$$

Figure 5 shows a comparison of the three activation functions defined above. One more important activation function is the Softmax function, typically used in the last layer of classification networks,

$$\mathrm{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n_m} e^{z_j}}. \tag{14}$$

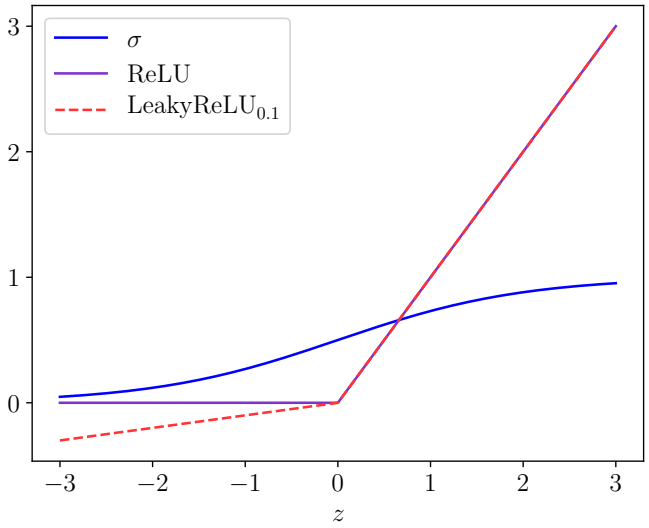

Figure 5: Comparison of three common activation functions in ANNs. See Eqs. (11), (12), and (13).

The function normalizes the activations in the layer to be non-negative and sum to 1, so that the activations represent a probability distribution over the classes.

The training of the parameters is accomplished with the use of gradient descent, as described in the previous section. Since the learned function is a composition, the gradient must be computed with the use of the chain rule. Let $\mathcal{J} = \mathcal{J}(X, Y; f_W)$ be the loss function. The first partial derivative is simply the partial derivative of the loss function with respect to the output of the network, $\partial\mathcal{J}/\partial f_W = \partial\mathcal{J}/\partial a^{(m)}$. For example, for the mean squared error in Eq. (7),

$$\frac{\partial \mathcal{J}}{\partial f_W} = \frac{2}{N}\sum_{i=1}^{N}(f_W(x_i) - y_i), \tag{15}$$

where $f_W(x_i)$ and y_i are to be understood as vectors. Now, consider that we know the derivatives with respect to activations in layer $l+1 \in \{2, \ldots, m\}$ and want to know the derivative with respect to activations in the previous layer l. The problem can be broken down with the chain rule into the part that is known, $\partial\mathcal{J}/\partial a^{(l+1)}$, and an unknown part $\partial a^{(l+1)}/\partial a^{(l)}$. Component-wise the partial derivative is

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial a_k^{(l)}} &= \sum_{p=1}^{n_{l+1}} \frac{\partial \mathcal{J}}{\partial a_p^{(l+1)}} \frac{\partial a_p^{(l+1)}}{\partial a_k^{(l)}} \\
&= \sum_{p=1}^{n_{l+1}} \frac{\partial \mathcal{J}}{\partial a_p^{(l+1)}} \frac{\partial g_{l+1}(W^{(l+1)}a^{(l)} + b^{(l+1)})_p}{\partial a_k^{(l)}} \\
&= \sum_{p=1}^{n_{l+1}} \frac{\partial \mathcal{J}}{\partial a_p^{(l+1)}} g'_{l+1}(z_p^{(l+1)}) W_{pk}^{(l+1)},
\end{aligned} \tag{16}$$

where $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$. Alternatively, in vectorized form

$$\frac{\partial \mathcal{J}}{\partial a^{(l)}} = \left(W^{(l+1)}\right)^T \left(\frac{\partial \mathcal{J}}{\partial a^{(l+1)}} \odot g'_{l+1}(z^{(l+1)})\right), \tag{17}$$

where $\odot$ denotes a component-wise product. The derivatives with respect to the weights in layer l can be calculated by invoking the chain rule once more,

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial W_{ij}^{(l)}} &= \sum_{k=1}^{n_l} \frac{\partial \mathcal{J}}{\partial a_k^{(l)}} \frac{\partial a_k^{(l)}}{\partial W_{ij}^{(l)}} \\
&= \sum_{k=1}^{n_l} \frac{\partial \mathcal{J}}{\partial a_k^{(l)}} \frac{\partial g_l(W^{(l)}a^{(l-1)} + b^{(l)})_k}{\partial W_{ij}^{(l)}} \\
&= \sum_{k=1}^{n_l} \frac{\partial \mathcal{J}}{\partial a_k^{(l)}} g'(z_k^{(l)}) \delta_{ik} a_j^{(l-1)} \\
&= \frac{\partial \mathcal{J}}{\partial a_i^{(l)}} g'(z_i^{(l)}) a_j^{(l-1)},
\end{aligned} \tag{18}$$

or in vectorized form

$$\frac{\partial \mathcal{J}}{\partial W^{(l)}} = \left(\frac{\partial \mathcal{J}}{\partial a^{(l)}} \odot g'(z^{(l)})\right) \left(a^{(l-1)}\right)^T. \tag{19}$$

Similarly, for the bias vector

$$\frac{\partial \mathcal{J}}{\partial b^{(l)}} = \frac{\partial \mathcal{J}}{\partial a^{(l)}} \odot g'(z^{(l)}). \tag{20}$$

All the derivatives can be calculated recursively in this manner, advancing backwards through the network. The intermediate results $a^{(l)}$ and $z^{(l)}$ do not need to be calculated again, since they can be stored on the forward pass of the network. This process of gradient calculation used in gradient descent in ANNs is referred to as backpropagation.

Many of the state-of-the-art ANN architectures have millions of parameters, which make them very expressive, but also prone to overfitting. A simple and often used form of regularization is dropout, which probabilistically ignores activations in a layer of the network during training [34]. This can be seen as sampling and training a larger set of thinned networks, which are averaged during test time.

One important theoretical result for ANNs is the Universal Approximation Theorem, which states that an ANN with a single hidden layer with a sufficient amount of neurons and a non-constant, continuous, and bounded activation function is able to approximate any continuous function over compact subsets of $\mathbb{R}^n$ [35]. The implication is that ANNs should, at least in theory, be able to solve most problems of practical relevance. However, contrary to the theorem, in practice it is found that having a deep network, rather than a wide one, is more useful. Furthermore, the often-used ReLU activation does not satisfy the boundedness property. Indeed, other theoretical results have proven the universal approximation property of width-bounded ReLU networks on Lebesgue integrable functions [36].

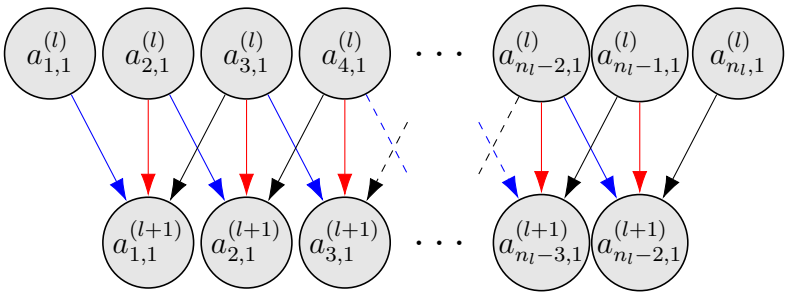

Figure 6: Illustration of connections in a 1D convolutional neural network layer with three weights and a single channel. Connections with matching colours indicate shared weights.

2.2.3 Convolutional neural networks

There are many learning tasks where the information in the input data is spatially ordered. For example, in images the order of the pixels relative to each other is important, but the precise location of the pixels is unimportant. This is problematic for the fully connected ANNs discussed above. If the inputs to the network in Fig. 4 were shifted by one neuron, the output of the network could completely change. This problem of translational invariance is solved by the convolutional neural network (CNN). In a CNN, the neurons are only locally connected to the next layer and the weights are shared across sets of neurons. This idea is illustrated in Fig. 6. In a fully connected network this corresponds to a sparse weight matrix, which is non-zero only close to the diagonal, and where the weights are the same in each row. This sparsity significantly reduces the number of parameters in the network, which also helps to reduce overfitting. For the connections in Fig. 6, the weight matrix of a fully connected layer would be

$$W^{(l)} = \begin{bmatrix} w_1 & w_2 & w_3 & & & & \\ & w_1 & w_2 & w_3 & & 0 & \\ & & & \ddots & & & \\ & 0 & & w_1 & w_2 & w_3 & \\ & & & & w_1 & w_2 & w_3 \end{bmatrix} \in \mathbb{R}^{(n_l-2)\times n_l}, \tag{21}$$

where $w_{1/2/3}$ correspond to the blue/red/black connections and n_l is the number of neurons in the layer l. Typically each layer in a CNN has multiple channels, each of which have their own set of weights. The weights of a CNN can be trained with backpropagation similar to regular ANNs.

In 1D, if the activations in an input layer are $a^{(l-1)}$, the weights of a filter are $w^{(l)}$, and $b^{(l)}$ is the bias vector, the output of a convolutional layer is

$$z^{(l)}_{i,c_2} = \sum_{c_1=1}^{n_\mathrm{c}} \sum_{p=0}^{n_\mathrm{f}-1} a^{(l-1)}_{i+p,c_1} w^{(l)}_{p,c_1,c_2} + b^{(l)}_{c_2}, \tag{22}$$

where l is the layer index, c_1 and c_2 are the channel indices in layers $l-1$ and l, respectively, $i \in \{1, ..., n_{l-1} - (n_\mathrm{f} - 1)\}$ is the neuron index, n_c is the number

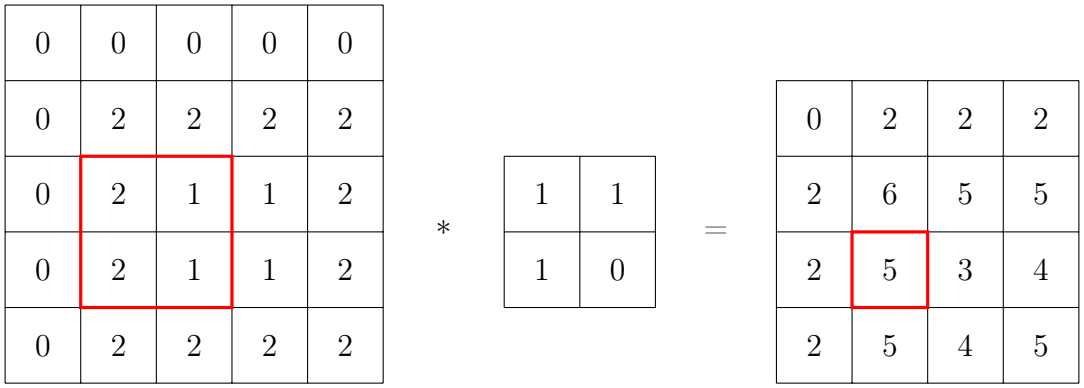

Figure 7: Example of a 2D convolution operation between a 4×4 pixel map and a 2×2 filter. The red rectangle indicates corresponding locations in the input and output. The input has been zero-padded on the top and left sides to preserve the same size in the output.

of channels in the layer, and n_f is the size of the filter. The number and size of the filters are hyperparameters of the model. Technically, this mathematical operation is cross-correlation, but it is called convolution by convention in deep learning context. As above, the activation in the following layer is obtained by applying the activation function, $a^{(l)}_{i,c_2} = g_l(z^{(l)}_{i,c_2})$.

The convolutional layer generalizes straightforwardly to any number of dimensions. In 2D and 3D,

$$\text{2D:} \quad z^{(l)}_{i,j,c_2} = \sum_{c_1=1}^{n_\mathrm{c}} \sum_{p=0}^{n_{\mathrm{f}_1}-1} \sum_{q=0}^{n_{\mathrm{f}_2}-1} a^{(l-1)}_{i+p,j+q,c_1} w^{(l)}_{p,q,c_1,c_2} + b^{(l)}_{c_2} \tag{23}$$

$$\text{3D:} \quad z^{(l)}_{i,j,k,c_2} = \sum_{c_1=1}^{n_\mathrm{c}} \sum_{p=0}^{n_{\mathrm{f}_1}-1} \sum_{q=0}^{n_{\mathrm{f}_2}-1} \sum_{r=0}^{n_{\mathrm{f}_3}-1} a^{(l-1)}_{i+p,j+q,k+r,c_1} w^{(l)}_{p,q,r,c_1,c_2} + b^{(l)}_{c_2}. \tag{24}$$

Figure 7 illustrates the effect in the 2D case. The operation can be seen as sliding a filter across a 2D pixel map, multiplying and summing the overlapping pixels to the output map at each location. The filter emphasizes certain features like edges in specific directions, which are reflected as larger activations in the output. Using multiple filters in each layer allows the network to encode the information in the input image in a rich set of features which can be used for learning complex mappings. Often it is useful to pad the input at the boundary to preserve the size of the feature map in the output. Possible padding schemes include zero-padding, reflective boundary, and periodic boundary.

While the convolutional layers respect translational invariance, CNNs often still have to terminate in fully connected layers, which do not have this property. A way to overcome this problem is to use pooling layers which summarise local information in the feature maps by down-sampling them. The two most common pooling operations are average pooling and max pooling, which are illustrated in Fig. 8 on a 2D pixel map. Average pooling takes the arithmetic mean of activations and max pooling takes the maximum of activations within local regions. In 2D, this can be

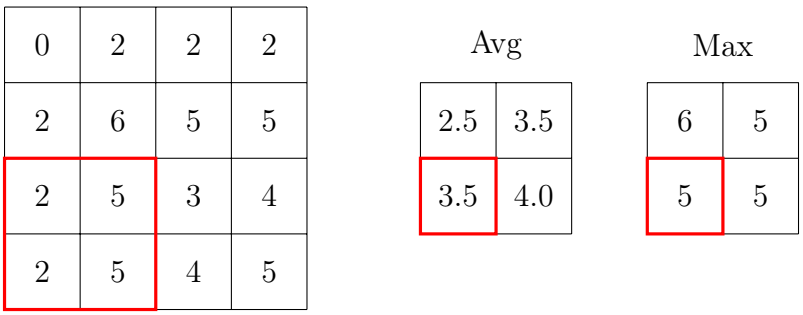

Figure 8: Example of the effect of average pooling and max pooling operations on a 4×4 pixel map with a 2×2 kernel and stride. The red rectangle indicates corresponding locations in the input and output.

viewed as a pooling kernel of size $k_\mathrm{w} \times k_\mathrm{h}$ sliding over an image in strides of $s_\mathrm{w} \times s_\mathrm{h}$, applying the pooling operation at each location and storing the result in a new pixel map. The resulting image has size that has been reduced by a factor of s_w in width and s_h in height. Usually $k_\mathrm{w} = s_\mathrm{w}$ and $k_\mathrm{h} = s_\mathrm{h}$ so that the regions of the pooling operations do not overlap. Pooling operations are applied several times at different parts of the network before the fully connected layers, so that the network becomes insensitive to local transformations in the input data. The lower resolution representations of the data encode the information in a more abstract way that is not so much dependent on the location of features. The down-sampling additionally has the benefit of allowing increasing the number of filters in deeper layers without significantly increasing the computational cost of training and evaluating the network.

Even though CNNs have significantly less parameters than fully connected networks, regularization may still be needed. In classification tasks, CNNs often terminate in fully connected layers, which can use dropout for regularization. However, dropout is usually not used in convolutional layers, where the zeroing of activations does not have a strong regularizing effect due to nearby activations being highly correlated. An alternative strategy called Spatial Dropout that randomly zeros out complete feature maps (channels) has been used in convolutional layers [37]. Other regularization methods for CNNs include adding random noise to the inputs and taking random crops, rotations, and flips of the input feature maps. These methods can be seen as augmenting the dataset to enforce invariances in the model. For example, a dog flipped left-to-right or viewed at different apparent sizes should still be recognized as a dog.

An early success for CNNs came in 1989 when LeCun et al. [38] applied back-propagation to a CNN for recognizing handwritten digits. Previous efforts had used hand-crafted weights in the convolutional layers. A more recent break-through in CNNs was AlexNet [11], which in 2012 won by significant margin the ImageNet Large Scale Visual Recognition Challenge, a competition for classifying over a million images into 1000 categories. AlexNet used what is now a fairly standard architecture for CNNs: 5 convolutional layers with 3 max-pooling layers and 3 fully connected layers with dropout regularization. Additionally, AlexNet leveraged the massive parallel computing capability of GPUs, which allowed the training of more

complicated models.

Depending on the task, it may not be necessary to flatten the spatially structured convolutional layers into fully connected layers. Fully convolutional networks have been used in, for example, image segmentation tasks [14, 15], where the goal is to segment an image based on categories of objects present in the image. These networks have an encoder-decoder type structure, where the image is first down-sampled with pooling layers and then up-sampled back to the original size. The representation of the data in the low-resolution part is often said to be in the latent space of the data. While the pooling is not required for enforcing translational invariance, it has the advantage of allowing an increased number of filters and enables the learning of correlations at multiple scales even with small filter kernel sizes.

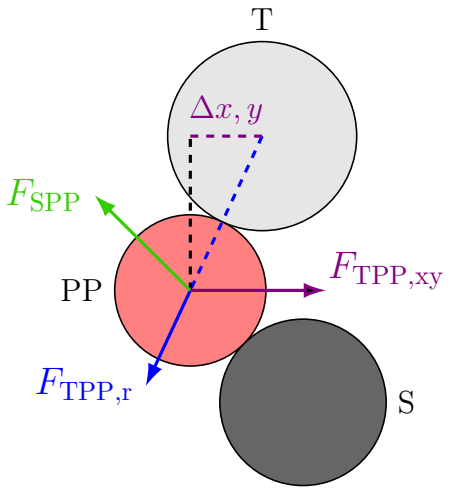

Figure 9: Illustration of the force components between the tip (T), the probe particle (PP) and the sample (S) in the Probe Particle Model. The S-PP force interaction F_{SPP} consist of Lennard-Jones and electrostatic force, and the T-PP force interaction consist of lateral and radial spring force components $F_{\mathrm{TPP,xy}}$ and $F_{\mathrm{TPP,r}}$, respectively.

3 Methods

This section explains in detail the methods used in the computational experiments of this thesis. Described are the simulation model for AFM simulations, the design of the descriptors, the details of the machine learning models, and the methods used in preprocessing of the data.

3.1 Atomic force microscope simulations

One challenge with high-resolution AFM is that it can take a long time, on the order of a full day, to get even a single good image of a sample. From a deep-learning perspective, this is a problem, because training a complex ANN with a large number of parameters requires a large number of training samples. Additionally, in experimental AFM, if the imaged sample structure is non-planar, the interpretation of the recorded signal is highly non-trivial, making accurate labelling by human labour impossible. These factors lead us into training our deep learning model with simulated AFM images.

The simulation model that we use is the Probe Particle Model [17, 18]. This is a mechanical model where a probe particle, modelling the flexible tip apex on the AFM tip, interacts with the sample through classical force fields. The interaction of the probe particle with the sample atoms is described by the Lennard-Jones potential, shown in Eq. (1), and the electrostatic interaction is described by Coulomb's law with partial point charges on all atoms. We use the Lennard-Jones parameters from the OPLS force field [39]. The interaction of the probe particle with the tip is described by a spring force with a separate spring constant in the lateral and radial directions. This idea is illustrated in Fig. 9. The tip is initially placed at some distance above the sample and it then approaches the sample at fixed steps in the

vertical direction. At each step, the probe particle is allowed to relax until all of the forces are in equilibrium. The sample plane is scanned by repeating this process at a grid of xy positions. The forces on the particle trajectories can be integrated with Giessibl's formula, shown in Eq. (5), to obtain the frequency shift at all heights. We use two probe particles in our experiments, CO and Xe. The differently charged probe particles result in different distortions of the atomic positions in the AFM image.

The model is implemented in OpenCL [40], enabling GPU acceleration of the code. As a result, we can simulate ~ 50 stacks (sets of constant-height) of AFM images per second, so that the training data can be generated on the fly while the deep learning model is training.

We do the simulations on a database of mostly organic molecular structures that are optimized with Density Functional Theory [41] to obtain the positions and partial charges of all atoms. The database has two distinct sets. The first set consists of ~ 134000 light molecules containing the elements H, N, C, O, and F. The second set consists of ~ 4300 heavier molecules that additionally contain Si, P, S, Cl, and Br. The molecules in the light set contain between 3 and 29 atoms, and the molecules in the heavy set contain between 2 and 47 atoms. Table 1 shows the distributions of elements in the two molecule datasets. In the light molecule dataset the elements H, N, C, and O are very common, and in the heavy molecule dataset, additionally S and Cl are commonly found, but the other elements, F, Si, P, and Br, are more uncommon.

The datasets are augmented by scanning each molecule from multiple directions. This is sensible, because AFM is sensitive only to the atoms closest to the probe tip. An asymmetric molecule rotated to a different configuration will appear completely different in the AFM image. We choose an initial set of 100 rotations from a roughly even distribution of points on a sphere and choose 20–30 best rotations, which maximize the function

$$S = \sum_{i=1}^{N_{\text{atoms}}} \exp(-\beta(z_i - z_{\text{close}})), \tag{25}$$

where N_{atoms} is the number of atoms in the molecule, z_i are the z-coordinates of the atoms in the scanning direction, z_{close} is the z-coordinate of the atom closest to the tip, and β is a decay parameter for which we choose the value $1.0\,\text{Å}^{-1}$. Choosing rotations with this criterion maximizes the number of atoms visible in the image, which helps to maximize the learning value of each training example.

3.2 Descriptors

Chemically, the features that most characterise a molecule are the relative positions and atomic numbers of the atoms that comprise the molecule. However, a simple list of atomic numbers and positions is not a suitable learning target for an ANN. If the molecule undergoes a rigid spatial translation or rotation, or the order of the atoms in the list is permuted, the prediction target becomes different even though the molecule itself has not changed. Furthermore, the varying number of atoms in molecules would make for a target of varying size, which presents a great difficulty in

Table 1: Statistics on the distribution of elements in the two molecule sets. Here, N_{mol} is the number of molecules that contain the given element in the given set. Multiple instances of the same element in the same molecule are not counted.

	Light set		Heavy set	
Element	N_{mol}	% of molecules	N_{mol}	% of molecules
H	133835	99.96	4205	97.4
N	82859	61.9	2251	52.1
C	133882	99.998	4246	98.4
O	113938	85.1	2734	63.3
F	2163	1.6	210	4.9
Si	0	0	127	2.9
P	0	0	375	8.7
S	0	0	1705	39.5
Cl	0	0	2122	49.2
Br	0	0	444	10.3
Total	133885	100.0	4317	100.0

the design of the model, although this has been overcome to an extent in recurrent neural networks.

Some descriptors, such as the Smooth Overlap of Atomic Orbitals [42], encode the chemical information in a way that respects the above invariances, but they are not suitable for our case, because they hide the original structural information. Instead, we have designed several descriptors which are intuitive to a human operator and are compatible with CNNs, which have an in-built translational invariance. In addition to the intrinsic invariances of the problem, we are constrained by the practical limitations of the experimental and simulation methods. The information should be encoded in a way that is robust to experimental noise and the imperfect simulation of reality. The descriptor should also not contain information that is not represented in the AFM image. Notably, atoms that are deep in the scanning direction contribute very little to the interaction with the probe and should not be represented in the descriptor.

- **Height Map**. An isosurface of the z component of the total force field around the molecule represented as a 2D pixel map. Contributions that are more than 2 Å below the maximum point are cut off. See Fig. 10a for an example. The Height Map corresponds to the height profile that would be obtained in constant-force operation mode with a rigid tip apex, which is not achievable in reality.

- **vdW Spheres**. Each atom is represented by a sphere with a radius equal to the van der Waals (vdW) radius of the atom. The value of a pixel is the height of the closest surface of a sphere in that location. The cut-off is 1.5 Å below the maximum point. Compared to the Height Map which is an aggregate of force contributions from multiple atoms, the vdW Spheres descriptor is better suited for identifying individual atoms, and is visually a very intuitive representation of the atomic structure. See Fig. 10b for an example.

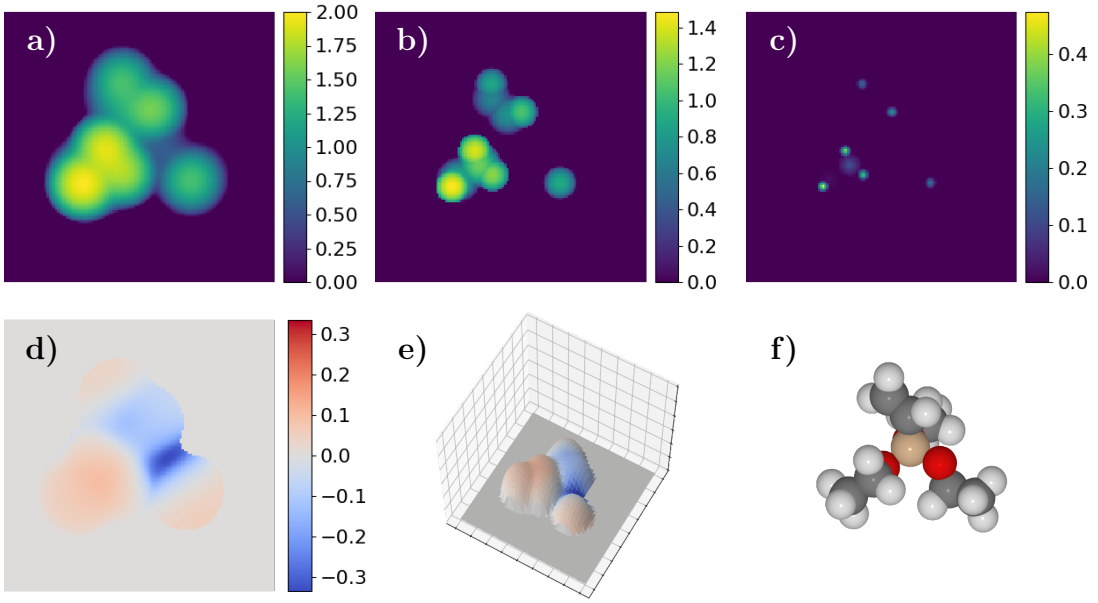

Figure 10: Examples of the descriptors that we use. a) Atomic Disks, b) vdW Spheres, c) Height Map, d) Electrostatic Map, e) combined Height Map and Electrostatic Map shown on a 3D surface, and f) the complete molecular structure from which the descriptors have been calculated. The size of the region in the descriptors is $16 \times 16\,\text{Å}^2$ discretized on a 128×128 pixel map.

- **Atomic Disks**. Each atom is represented by a disk that decays conically away from the center position of the atom in the xy-plane. The width of the cone is proportional to the covalent radius of the atom, and the value at the tip of the cone indicates the height of the atom. Atoms with nucleus more than 1 Å below the nucleus of the top atom are cut off. This is a more abstract descriptor that pinpoints the exact locations of the atoms more precisely than the vdW Spheres descriptor. See Fig. 10c for an example.

- **Electrostatic Map**. The electrostatic potential of the molecule is calculated on the surface corresponding to the Height Map descriptor for a 3D representation of the charge distribution over the molecule. The electrostatic potential is calculated as a sum of point charge potentials based on partial charges on each atom. This descriptor is interesting in terms of the chemical properties of the molecule and may help in the identification of atomic species. See Figs. 10d and 10e for examples.

- **xyz**. The above descriptors provide information on the structure of the molecule, but they are not readily translatable back to a list of atom coordinates, and due to the reasons discussed above, a list of atom coordinates has many issues as a learning target. The xyz descriptor sidesteps these issues by using a representation of default grid positions for the objects to be identified, an idea that has been used in learning bounding boxes of objects in computer vision [16]. Each atom is assigned to one box on the grid and its position is expressed as an offset from the default position at the center of that

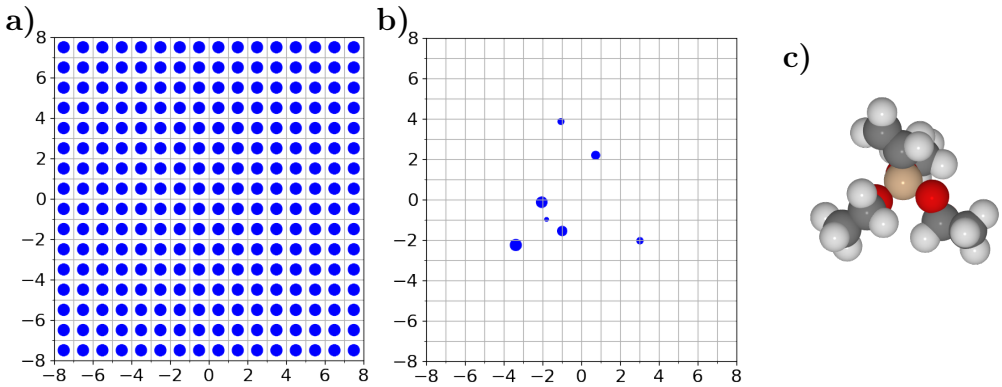

Figure 11: xyz descriptor. a) A grid of default positions at the centres of the square boxes. b) An assignment of atoms into the boxes. The sizes of the circles denote the relative heights of the atoms. c) The molecular geometry from which the descriptor is calculated.

box. If multiple atoms would be assigned to the same box, the atom with the highest z-coordinate is chosen. The zero point in z-direction is at the nucleus of the highest atom, and those atoms whose nuclei are below $-0.7\,\text{Å}$ are cut off. See Fig. 11 for a visual illustration of the idea. Each box on the grid has an associated one-hot encoded class vector with the classes *background* and *atom* based on whether an atom is assigned to the box, and if the class is *atom*, it has the off-set xyz-coordinates of the atom. The learning task, therefore, has two parts: a classification task for the presence of atoms and a regression task for the positions of the atoms. A prediction of this descriptor assigns each box a confidence level on the classes, so that a list of atom coordinates can be retrieved by taking the off-set coordinates of those grid positions with confidence level on the *atom*-class above some threshold.

3.3 Machine learning models

The design of deep learning architectures is highly empirical in nature. This makes it difficult to point out the exact purpose or effect that a particular component of a network has. From a physics perspective it is somewhat unsatisfactory that it cannot be said what the physical meanings of the parameters are or if there is even any. However, there are some intuitions and practical concerns governing the design decisions.

- **Learning 3D spatial correlations.** The models feature 3D convolution layers, which are necessitated by the 3D nature of the input data. If only 2D convolution layers were used, the correlations in the z-direction would be immediately lost in the first layer. 3D CNNs have been considered difficult due to being computationally expensive, but recently they have been proven useful in learning spatiotemporal features [43] and in real-time object recognition [44].

- **Avoiding representational bottlenecks.** In the middle of the network, avoiding layers with too high compression has been deemed a good principle in the design of CNNs [13]. This principle leads us to choosing an increasing number of filters with the decreasing size of the feature maps. On the other hand, having too many parameters easily leads to overfitting, so a balance must be found. In our case this is especially important, since we train on simulated data but want to generalize to experimental data.

- **Keeping the model computationally tractable.** In our initial attempts, we tried to maintain the xy-dimension of the feature maps throughout the network. However, especially the 3D convolution layers are very expensive in both computation time and memory consumption, which made the training very slow or even impossible with even relative shallow networks. To mitigate this problem, we instead opt to quickly downsample the feature maps with pooling layers and then upsample them at the end of the network to match the original size. The intuitive picture is that the information in the input AFM image is encoded into an abstract representation that is then decoded into the desired descriptors.

All of the models used here are fully-convolutional neural networks. We use three models for three different learning tasks: firstly, the DSH (Disks-Spheres-Height) model for simultaneously predicting the Atomic Disks, vdW-Spheres, and Height Map descriptors, secondly, the ES (ElectroStatics) model for simultaneously predicting the Electrostatic Map and the Height Map, and lastly, the xyz model for predicting the xyz descriptor. See Fig. 12 for illustrations of the model structures. The models are implemented in Keras [45] with TensorFlow [46] backend.

The DSH and ES models have an encoder-decoder type architecture. Tables 2 and 3 lists all the layers of these networks. The encoder and decoder parts are the same for both models. The major differences are in the number of input and output branches and the layers in the latent space parts of the models.

The encoder consists of three 3D convolution layers with increasing number of filters from 4 to 8 to 16. Each 3D convolution is followed by an average pooling layer with kernel size $2 \times 2 \times 2$, except the last one, which has kernel size $2 \times 2 \times 1$. We found not reducing the size in the z-direction in the last pooling layer to yield better performance, possibly because reducing the size too much would result in an informational bottleneck. Both average and max pooling were tried, but we found average pooling to work better. The stride for all pooling layers is equal to the kernel size so that the pooling operations are non-overlapping.

In the decoder the feature maps are up-sampled in three stages mirroring the down-sampling stages. Each up-sampling stage consists of a nearest neighbour interpolation followed by a pair of 2D convolutions with 16 filters. We also tried transposed convolutions for the up-sampling operation. The transposed convolution, also known as deconvolution, in a sense reverses the effect of a regular convolution [47]. However, we found the output from the model to contain patterned artefacts, which are a known to be a problem with transposed convolutions [48].

All the convolutional layers except the output layers use a $\text{LeakyReLU}_{0.1}$ activation. We tried using ReLU activations, but this resulted in many layers having

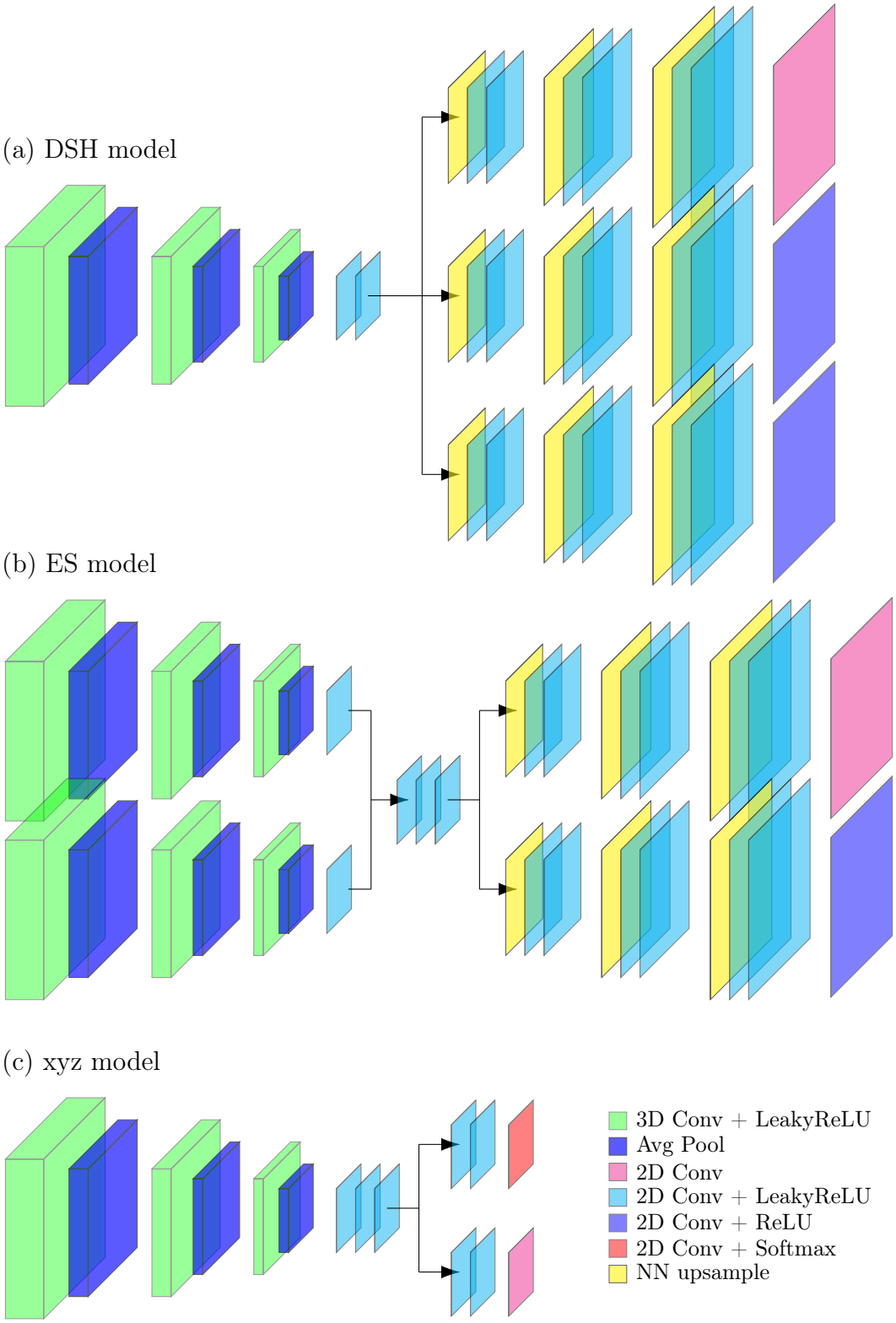

Figure 12: Schematic illustrations of the model architectures. The forward direction is from left to right. The sizes of the layers represent the relative size of the feature maps. Not to scale.

Table 2: DSH model architecture. The factors in parentheses denote parallel layers for separate output branches. The total number of parameters is 122,811.

	Layer type	Output dimension	Kernel size	Activation	Parameters
0	Input	$128 \times 128 \times 10 \times 1$	-	-	-
1	3D conv	$128 \times 128 \times 10 \times 4$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	112
2	Avg pool	$64 \times 64 \times 5 \times 4$	$2 \times 2 \times 2$	-	-
3	3D conv	$64 \times 64 \times 2 \times 8$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	872
4	Avg pool	$32 \times 32 \times 2 \times 8$	$2 \times 2 \times 2$	-	-
5	3D conv	$32 \times 32 \times 2 \times 16$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	3472
6	Avg pool	$16 \times 16 \times 2 \times 16$	$2 \times 2 \times 1$	-	-
7	Reshape to 2D	$16 \times 16 \times 32$	-	-	-
8	2D conv	$16 \times 16 \times 64$	3×3	$\text{LeakyReLU}_{0.1}$	18496
9	2D conv	$16 \times 16 \times 64$	3×3	$\text{LeakyReLU}_{0.1}$	36928
10	NN-upsample	$32 \times 32 \times 64(\times 3)$	-	-	-
11	2D conv	$32 \times 32 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$9232(\times 3)$
12	2D conv	$32 \times 32 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 3)$
13	NN-upsample	$64 \times 64 \times 16(\times 3)$	-	-	-
14	2D conv	$64 \times 64 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 3)$
15	2D conv	$64 \times 64 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 3)$
16	NN-upsample	$128 \times 128 \times 16(\times 3)$	-	-	-
17	2D conv	$128 \times 128 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 3)$
18	2D conv	$128 \times 128 \times 16(\times 3)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 3)$
19	2D conv	$128 \times 128 \times 1(\times 3)$	3×3	None/ReLU/ReLU	$145(\times 3)$

Table 3: ES model architecture. The factors in parentheses denote parallel layers for separate input/output branches. The total number of parameters is 169,970

	Layer type	Output dimension	Kernel size	Activation	Parameters
0	Input	$128 \times 128 \times 10 \times 1(\times 2)$	-	-	-
1	3D conv	$128 \times 128 \times 10 \times 4(\times 2)$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	$112(\times 2)$
2	Avg pool	$64 \times 64 \times 5 \times 4(\times 2)$	$2 \times 2 \times 2$	-	-
3	3D conv	$64 \times 64 \times 2 \times 8(\times 2)$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	$872(\times 2)$
4	Avg pool	$32 \times 32 \times 2 \times 8(\times 2)$	$2 \times 2 \times 2$	-	-
5	3D conv	$32 \times 32 \times 2 \times 16(\times 2)$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	$3472(\times 2)$
6	Avg pool	$16 \times 16 \times 2 \times 16(\times 2)$	$2 \times 2 \times 1$	-	-
7	Reshape to 2D	$16 \times 16 \times 32(\times 2)$	-	-	-
8	2D conv	$16 \times 16 \times 64(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$18496(\times 2)$
9	Concatenate	$16 \times 16 \times 128$	-	-	-
10	2D conv	$16 \times 16 \times 64$	1×1	$\text{LeakyReLU}_{0.1}$	8256
11	2D conv	$16 \times 16 \times 64$	3×3	$\text{LeakyReLU}_{0.1}$	36928
12	2D conv	$16 \times 16 \times 64$	3×3	$\text{LeakyReLU}_{0.1}$	36928
13	NN-upsample	$32 \times 32 \times 64(\times 2)$	-	-	-
14	2D conv	$32 \times 32 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$9232(\times 2)$
15	2D conv	$32 \times 32 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 2)$
16	NN-upsample	$64 \times 64 \times 16(\times 2)$	-	-	-
17	2D conv	$64 \times 64 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 2)$
18	2D conv	$64 \times 64 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 2)$
19	NN-upsample	$128 \times 128 \times 16(\times 2)$	-	-	-
20	2D conv	$128 \times 128 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 2)$
21	2D conv	$128 \times 128 \times 16(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$2320(\times 2)$
22	2D conv	$128 \times 128 \times 1(\times 2)$	3×3	None/ReLU	$145(\times 2)$

"dead" channels that have zero or very minimal noisy output, and we found that LeakyReLU yielded better results in practice. All the convolutional layers have reflective padding such that the size of the feature map is preserved. The stride for all convolutional layers is 1 and the kernel size is 3×3 for 2D convolutions and $3 \times 3 \times 3$ for 3D convolutions unless otherwise noted.

The DSH model starts with the encoder, after which the 16-channel 3D feature maps with 2 z-slices are reshaped into 32 channels of 2D feature maps. This is followed by a pair of 2D convolutions with 64 filters. The small feature maps of the latent space are then decoded in three separate branches of the decoder, each with independent weights and outputting a different descriptor. At the end of each decoder branch there is one more 2D convolution for reducing the channel size to one. The vdW Spheres and Height Map branches end in ReLU activations, which is a natural choice, since the output feature maps have zero-point at the cutoff height and all the relevant features should be positive. However, the Atomic Disks branch has no activation, because we found that with ReLU activations the model would learn to predict only zeros. This is likely caused by the fact that the Atomic Disk descriptor of most structures is quite sparse. Even an all-zero output has fairly low loss, and the gradient of ReLU is zero on the negative side, so the weights never get updated in favour of a non-zero prediction.

The ES model features two different input branches, each of which receive a different channel of information on the imaged molecule structure. Each input branch consist of the 3D encoder followed by a 2D convolution with 64 filters. The two input branches are combined by concatenating the feature maps in the channel dimension for a layer with a total of 128 channels. After the concatenation, there is a 2D convolution layer with a 1×1 kernel. This has the function of reducing the number of parameters and the computational cost of the following layer. The rest of the model architecture is the same as in the DSH model except with two decoder branches. The last layer in the branch for the Electrostatic Map prediction has no activation function since the values can be both negative and positive, and the Height Map prediction has a ReLU activation.

The two input branches of the ES model take as input AFM images imaged with two differently functionalized tips, one with CO (negative), and one with Xe (positive). The two differently charged tips present two different views of the same system [49]. Since the different distortions in the images result explicitly from the difference in electrostatic interaction, this should be very useful in helping the model differentiate the effect of the electrostatic potential from other interactions. We initially tried to predict the Electrostatic Map descriptor from just a single AFM image with a CO-tip, using a model similar to the DSH model with two outputs, but found that the results were not very accurate or reliable.

The loss functions for the DSH and ES models are weighted sums of the mean squared losses of the output feature maps averaged over the pixels. The weights for the DSH model are 20, 0.3, and 0.2 for Atomic Disks, vdW Spheres, and Height Maps, respectively, and weights for the ES model are 1.0 and 0.1 for the Electrostatic Map and Height Map, respectively. The weights are necessary for balancing the contributions from each output to the total loss. The magnitudes of the losses are different due to the different scales in the descriptors. The values for the weights

Table 4: xyz model architecture. The factors in parentheses and the two parameter numbers in the last layer denote parallel layers for separate output branches. The total number of parameters is 560,941.

	Layer type	Output dimension	Kernel size	Activation	Parameters
0	Input	$128 \times 128 \times 10 \times 1$	-	-	-
1	3D conv	$128 \times 128 \times 10 \times 4$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	112
2	Avg pool	$64 \times 64 \times 5 \times 4$	$2 \times 2 \times 2$	-	-
3	3D conv	$64 \times 64 \times 2 \times 8$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	872
4	Avg pool	$32 \times 32 \times 2 \times 8$	$2 \times 2 \times 2$	-	-
5	3D conv	$32 \times 32 \times 2 \times 16$	$3 \times 3 \times 3$	$\text{LeakyReLU}_{0.1}$	3472
6	Avg pool	$16 \times 16 \times 2 \times 16$	$2 \times 2 \times 1$	-	-
7	Reshape to 2D	$16 \times 16 \times 32$	-	-	-
8	2D conv	$16 \times 16 \times 128$	3×3	$\text{LeakyReLU}_{0.1}$	36992
9	2D conv	$16 \times 16 \times 128$	3×3	$\text{LeakyReLU}_{0.1}$	147584
10	2D conv	$16 \times 16 \times 128$	3×3	$\text{LeakyReLU}_{0.1}$	147584
11	2D conv	$16 \times 16 \times 64(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$73792(\times 2)$
12	2D conv	$16 \times 16 \times 64(\times 2)$	3×3	$\text{LeakyReLU}_{0.1}$	$36928(\times 2)$
13	2D conv	$16 \times 16 \times (3/2)$	3×3	None/Softmax	1731/1154

were found through empirical testing, trying to balance the importance of each output.

The details for the layers of the xyz model are listed in Table 4. The model starts with the encoder as in the other models, but lacks the decoder part, since the grid of the xyz descriptor is in the coarse resolution of the latent space. The decoder is followed by three 2D convolutions with 128 filters. After this the network splits into two branches, one for the classification task and one for the regression task. Both branches have two 2D convolutions with 64 filters and one more 2D convolution with 2 filters and Softmax activation in the classification branch and 3 filters with no activation in the regression branch.

The loss function for the xyz model is considerably more complicated than with the other two models. The loss for the classification task is Focal Loss [50], which is a modified version of cross-entropy:

$$\text{FL}(p,t) = -\sum_{i=1}^{N_\text{grid}} \sum_{j=1}^{N_\text{class}} \alpha_i (1-p_{ij})^\gamma t_{ij} \log(p_{ij}), \tag{26}$$

where p is the prediction, t is the ground truth, N_grid is the number of grid points, $N_\text{class} = 2$ is the number of classes, γ is a hyperparameter and

$$\alpha_i = \begin{cases} \alpha_\text{n} & \text{if } t_{i1} = 0 \ (\textit{background}) \\ 1-\alpha_\text{n} & \text{if } t_{i1} = 1 \ (\textit{atom}), \end{cases} \tag{27}$$

where α_n is another hyperparameter. The ground truth vectors for each grid position are one-hot encoded, meaning that the value at the position of the true class is equal to 1 and the values for the other classes are 0. The *atom* class corresponds to $j = 1$ and the *background* class corresponds to $j = 2$. This means that the second sum in Eq. (26) picks out the loss only for the correct class at each position.

The role of the factor γ is to make the loss focus more on those predictions that are the most wrong, i.e., $(1 - p_{ij})$ close to 1. Larger values of γ lead to stronger focus. The parameter α_n serves to bias the loss more towards the positive examples (*atom*) than the negative ones (*background*). Considering that the grid size is $16 \times 16 = 256$ and typically less than 10 atoms are present in an image, there is a large class imbalance. The biasing factor α_n helps to balance the loss.

The loss on the position prediction is mean squared distance with an additional factor for discounting predictions on lower lying atoms

$$\mathrm{PL}(r, q) = \frac{1}{N_\mathrm{p}} \sum_{i=1}^{N_\mathrm{grid}} d_i t_{i1} [(r_{i\mathrm{x}} - q_{i\mathrm{x}})^2 + (r_{i\mathrm{y}} - q_{i\mathrm{y}})^2 + (r_{i\mathrm{z}} - q_{i\mathrm{z}})^2], \tag{28}$$

where r are the predicted position off-sets, q are the true position off-sets, $N_\mathrm{p} = \sum_i t_{i1}$ is the true number of atoms assigned to the grid, and

$$d_i = 1 - \frac{q_{i\mathrm{z}}}{2 z_\mathrm{min}}, \tag{29}$$

where $z_\mathrm{min} = -0.7\,\text{Å}$ is the cut-off height for the atoms. We only train on examples where $N_\mathrm{p} > 0$. In Eq. (28) the factor t_{i1} ensures that the loss is summed only over those positions where there is a reference atom position. The motivation for the scaling factor d_i is to penalize the model less for incorrectly predicting the position of those atoms which are less visible. There is naturally more uncertainty in the locations of those atoms due to noise and low signal level.

The losses for the classification and position prediction are combined to a total loss as

$$J(p, t, r, q) = \mathrm{FL}(p, t) + \beta_\mathrm{PL} \mathrm{PL}(r, q), \tag{30}$$

where β_PL is a hyperparameter for balancing the two loss terms. We set the hyperparameter values to $\gamma = 1.25$, $\alpha_\mathrm{n} = 0.30$, and $\beta_\mathrm{PL} = 0.15$. The values for α_n and β_PL were found through iterative testing, using intuition and trying to reach balanced behaviour of the model. The best value for γ was found by testing the model performance for the values $\gamma \in \{1.0, 1.25, 1.5, 1.75, 2.0\}$ while holding the other hyperparameters constant. The search is not meant to be exhaustive, as the problem of optimizing costly functions is in general a difficult one and beyond the scope of this thesis.

The performance of the xyz model can be evaluated by methods other than the loss function. The figures of merit for the classification task are the precision and recall,

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{31}$$

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{32}$$

where TP = True Positive is the number of positive examples correctly classified, and FN = False Negative and FP = False Positive are the number of positive and negative examples, respectively, incorrectly classified. The precision measures how

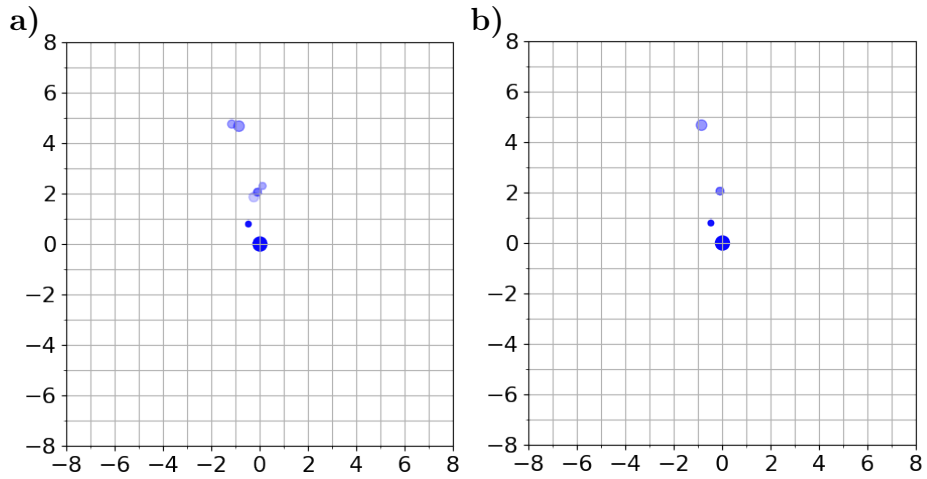

Figure 13: Example of the effect of non-maximum suppression. The prediction in **a)** has overlapping atoms, which have been removed in **b)**. The sizes of the circles denote the depth of the atoms, and the transparency of the circles denote the confidence levels.

often the predicted atoms from the model are truly there, and the recall measures how often the atoms that should have been found were found. In a good model, these two values should be reasonably balanced. It should be noted that the classification accuracy of the model is not a good measure of its performance due to the imbalance in class distribution. For example, even if the model always predicted *background*, the accuracy would still be over 95%, because more than 95% of the grid positions are in the class *background*.

If the true position of an atom is close to the border between boxes of the grid, the xyz model often tries to predict that atom from multiple boxes. In those cases, the prediction can contain multiple predicted atoms on top of each other. We can remove those atoms systematically by suppressing predictions that are closer than some distance R_{nms} to each other such that the predicted atoms with lower confidence levels are removed. This idea of non-maximum suppression has been used in the bounding box problem as well [16]. An example of this is shown in Fig. 13. In principle, two atoms should never be closer than 0.74 Å, the length of the H-H bond [51], but due to uncertainty in the position prediction, we set $R_{\text{nms}} = 0.5$ Å by default. However, since this is a post-processing step, the value could be adjusted freely to find the best value for each prediction using physical intuition.

With all models, the optimizer for gradient descent is the Adaptive Moment Estimation (Adam) [52] optimizer. We set the learning rate to 0.001, set the decay to 10^{-5}, and use the default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the moment decay parameters.

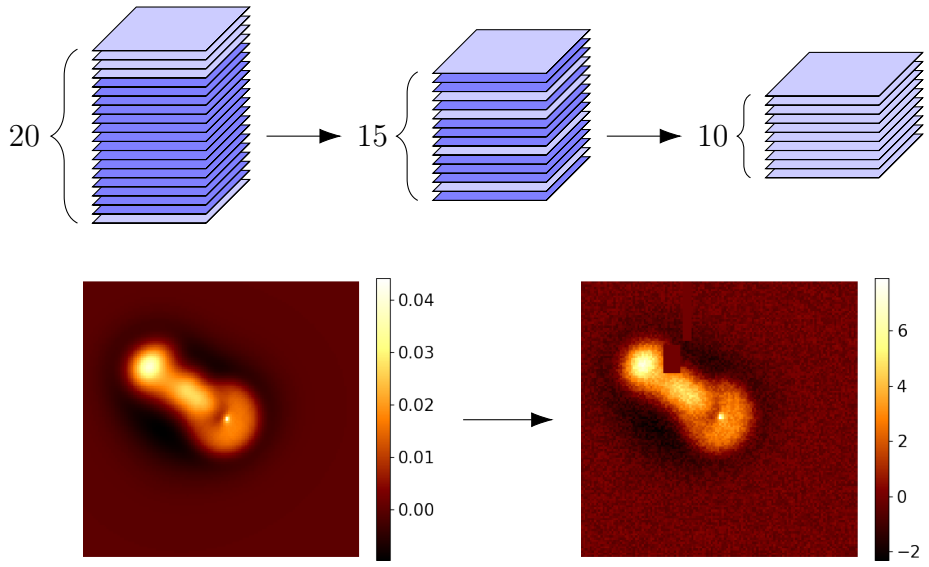

Figure 14: Illustration of preprocessing steps on simulated AFM stacks. Top: scanning height randomization. Bottom left: A simulated AFM image before the preprocessing steps. Bottom right: The same image processed with normalization, noise, pixel shift, and cutouts. See text for explanations.

3.4 Data processing and regularization

In the simulations the equilibrium tilt of the probe particle on the tip is varied randomly within a circle of radius 1.0 Å. This reflects the asymmetric adsorption of the particle functionalizing the tip that is typical in experiments. After the AFM image is obtained from the simulation, the image stack is processed in several ways for regularization and improved training. Figure 14 illustrates the effects of the pre-processing steps. The pre-processing steps in the order they are applied are

1. **Scan height randomization.** During experiments, the distance between the tip and the sample often cannot be known accurately, and there may be some variation in the heights where the image stack is obtained. To make the model insensitive to such variations, we randomize the AFM stack heights as a pre-processing step. The simulation scan always contains 20 slices at predetermined distances from the closest atom in the scan. From these 20 slices we first choose randomly a continuous stack of 15 z-slices to reflect possible variations in the tip-sample distance, and from these 15 we then choose randomly 10 to reflect lost scanning heights. The first step ensures that there are not too big jumps between adjacent heights in the final stack.

2. **Normalization.** Each height slice in the AFM stack is normalized by subtracting the mean and dividing by the sample standard deviation of the slice. Normalizing inputs has been generally found to be essential in many machine learning applications.

3. **Random noise.** Uniform random noise in range $[-A_z, A_z]$ is added to each

voxel of the image. The amplitude is

$$A_z = 0.5c(v_{\max} - v_{\min}), \tag{33}$$

where $c = 0.1$, and $v_{\min/\max}$ are the minimum and maximum pixel values in the corresponding z-slice. The value of c was chosen to visually reflect the degree of noise that is present in experimental AFM images.

4. **Random pixel shifts.** Each z-slice is randomly shifted by at most three pixels in the x- and y-directions. During experiments, it is common for there to be a slight drift in the xy-plane between the scanning of different heights. For this reason, the different slices need to be aligned before giving them as an input to the deep learning model, but the alignment process in not always accurate.

5. **Cutouts.** Random rectangular regions in each slice are cut out by setting them to zero. This kind of regularization method has been shown to improve the performance of CNNs in image classification tasks [53, 54]. Additionally, the cutouts bear resemblance to the kind of elongated artefacts, which are sometimes observed in experimental AFM images.

4 Results

The models are trained on a dual-GPU setup, where one GPU produces the simulated AFM training data and the other trains the CNN model weights. The training of one model takes roughly two days on a pair of Nvidia Tesla V100 GPUs. We train the models on two datasets containing molecules with different elements, as shown

Table 5: The number of molecules used in the training, validation, and test sets of the two datasets used to train and evaluate the models. See Table 1 for more details on the datasets.

	Dataset 1			Dataset 2		
Molecule set	Train	Validation	Test	Train	Validation	Test
Light set	5000	1500	2500	3500	900	1200
Heavy set	0	0	0	2500	600	1200
Total	5000	1500	2500	6000	1500	2400

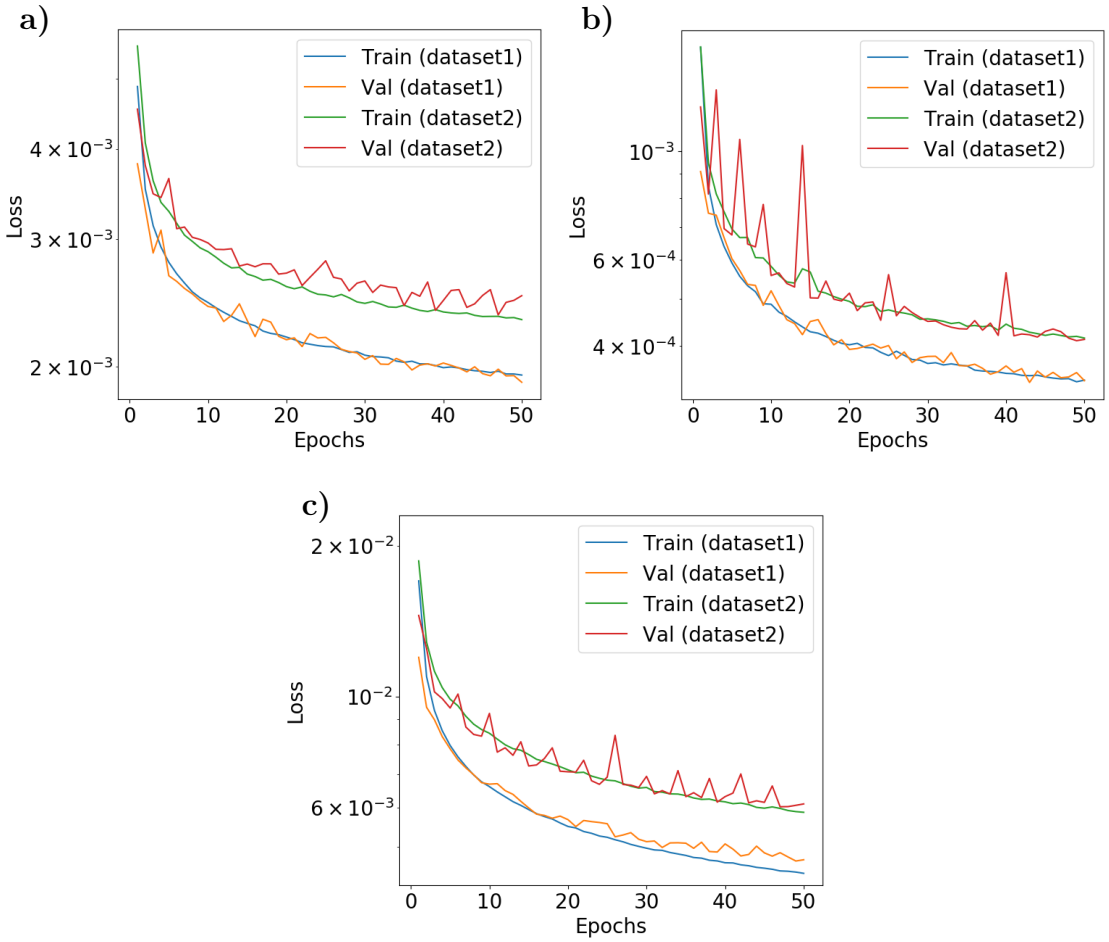

Figure 15: The training and validation losses of the models as functions of the number of training epochs on the two datasets for **a)** the DSH model, **b)** the ES model, and **c)** the xyz model.

Table 6: Losses on the training, validation, and test sets for all the models after the last epoch.

	Dataset 1			Dataset 2		
Model	Train	Validation	Test	Train	Validation	Test
DSH model	1.95×10^{-3}	1.90×10^{-3}	1.92×10^{-3}	2.32×10^{-3}	2.51×10^{-3}	2.59×10^{-3}
ES model	3.41×10^{-4}	3.40×10^{-4}	3.40×10^{-4}	4.15×10^{-4}	4.13×10^{-4}	4.26×10^{-4}
xyz model	4.43×10^{-3}	4.72×10^{-3}	4.72×10^{-3}	5.87×10^{-3}	6.10×10^{-3}	6.58×10^{-3}

in Table 5. Dataset 1 contains only light elements and dataset 2 is extended to additionally contain molecules with heavier elements. For the DSH and ES models we use 20 rotations per molecule and for the xyz model we use 30 rotations per molecule.

Figure 15 shows the training curves for all the models and Table 6 lists the final losses on both datasets. The loss is not significantly improving at around 50 epochs, where we choose to stop the training. A comparison of the training and validation losses suggests that there is a slight overfit on dataset 2 for the DSH model and on both dataset for the xyz model. The loss is higher on dataset 2 than dataset 1 for all models. This is likely in part due to larger variation of structures in dataset 2, and partially due to the molecules being on average larger and thus having more atoms present in the descriptors in dataset 2. More detailed results for each model are presented in the following sections.

4.1 DSH model

Figure 16 shows a few comparisons between reference solutions and predictions from the DSH model on simulated AFM images of molecules from the test sets. The predictions in Figs. 16a, 16b, and 16c represent the worst-, an average-, and the best-case predictions from the model based on the value of the loss function. Even in the worst case the model is accurate in locating where in the plane of the image the relevant features are. In this case, likely a mischaracterization of the bromine, which is quite rare in the training set, causes the model to misjudge the relative depths of some of the other atoms. Generally speaking, top-most atoms are clearly distinguishable, but the lower lying atoms appear more unclear and the prediction is generally more blurred than the reference. The prediction in Fig. 16d showcases an interesting scenario where our method presents its utility. Even an expert in the field would have very hard time correctly interpreting the AFM image, but the prediction, although somewhat blurred, clearly shows the carbon ring structure of the molecule.

Directly evaluating the performance of the models on experimental data is not yet possible, since the correct solution is not known *a priori*. What we do instead, is to make predictions on experimental data of a known molecule and compare the predictions with those from simulations, attempting to find a match. We do simulations of several hundred uniformly distributed rotations of the known molecule and match each experimental image to one of the simulated configurations by finding the pairing with highest cross-correlation between the predicted vdW Spheres descriptors. The advantage of comparing the predicted descriptors instead of the

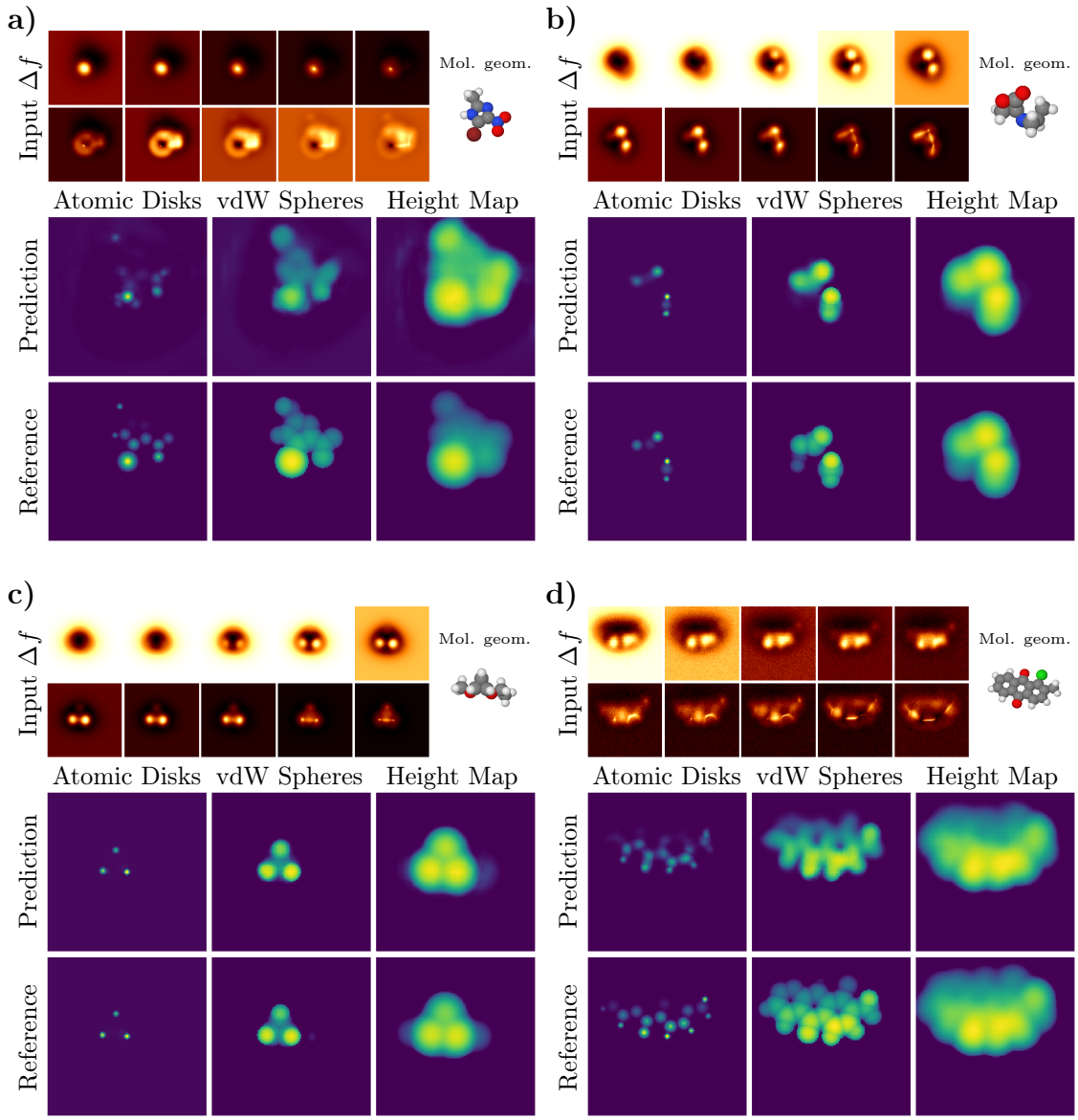

Figure 16: Four example predictions of the Atomic Disk, vdW Spheres, and Height Map descriptors on simulated data with the DSH model. The loss function is decreasing from **a)** to **c)**.

AFM images is that the descriptor is not dependent on the exact scanning heights of the AFM image stack and it should be robust against some of the noise that is present in the experiments.

We have experimental AFM data of 1S-camphor imaged on a Cu(111) surface in several configurations. Figure 17 shows the predictions of the vdW-Spheres descriptor on the experimental data along with the matching predictions from the simulations. The predictions on the experimental data show some scattered artefacts around the main atomic features, but otherwise the match with the predictions from the simulated data is qualitatively good. A qualitative comparison of the AFM images also confirms the plausibility of the configurations at least in the first four ex-

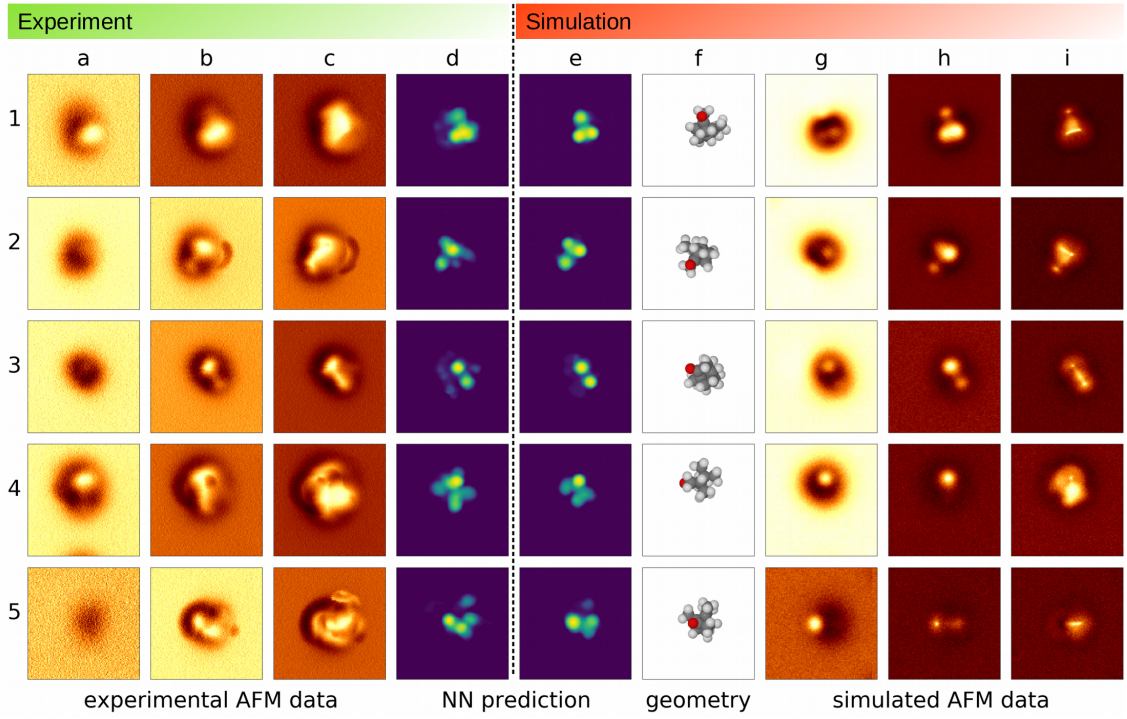

Figure 17: Matching between experimental and simulated AFM images of 1S-camphor. Each row corresponds to one experimental configuration. Columns a, b, and c show the experimental AFM images at far, middle, and close distance, respectively, and similarly, columns g, h, and i show the simulated AFM images. Only three out of the ten total slices fed to the model are shown. Columns d and e shown the predicted vdW Spheres descriptors for the experiment and simulation, respectively. Column f shows the molecular geometries corresponding to the simulations. Also published in a preprint in Ref. [55].

periments. In the fifth experiment some of the features present in the experimental AFM image are clearly absent in the simulated one. See discussion in Sec. 5.

4.2 ES model

The performance of the ES model on simulated data is generally very good. Figure 18a shows an example prediction that represents a roughly average performance. The accuracy of the Height Map is very good, with only some details in the deeper region being slightly blurry, similar to the DSH model. The accuracy of the Electrostatic Map is similarly good, qualitatively capturing very well the differences between regions of positive and negative charge and being almost spot-on everywhere except in the deepest parts. The prediction in Fig. 18b is the prediction with the highest loss in the test set of dataset 1. The high loss in this example is caused by the model incorrectly predicting the features that are located deeper in the scanning direction. However, even in this case, the prediction of the Electrostatic Map is qualitatively quite good, since the sign of the charge is correctly predicted in most parts of the image, although sometimes at the wrong absolute value.

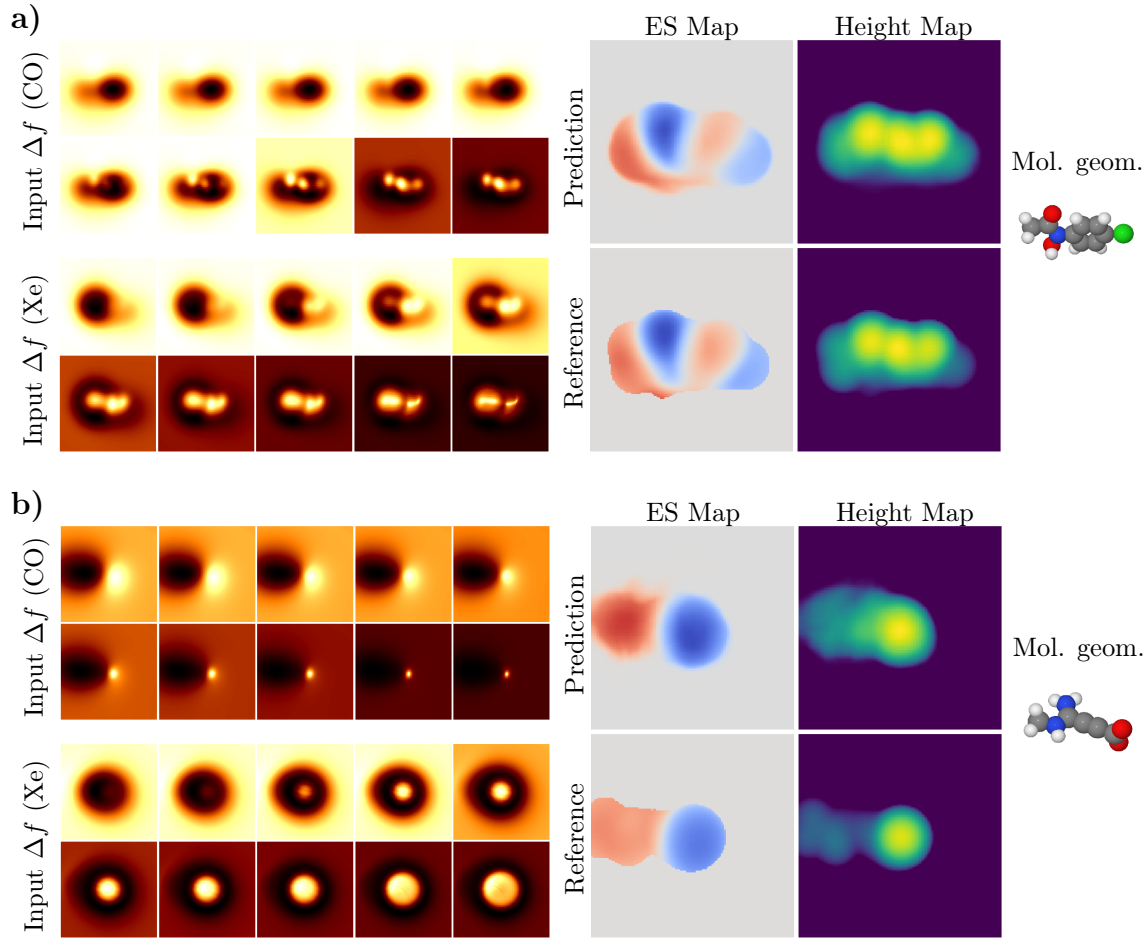

Figure 18: Two example predictions of the Electrostatic Map and Height Map descriptors from the ES model on simulated AFM images. The predictions represent roughly average and worst-case performance of the model.

It should be noted that there is an experimental challenge here. In the simulations, we are free to assume that the system imaged with the two different tips remains in exactly the same configuration and alignment. This is not a trivial thing to achieve in a real-world experiment. As such, we do not currently have the data to evaluate the model performance on experiments.

4.3 xyz model

The xyz model reaches quite a good level of performance on the simulated data. The model generally finds most of the top atoms with high degree of accuracy and there is generally more uncertainty in the deeper atoms as shown in the example prediction in Fig. 19a. In this prediction, one of the deeper atoms is missed and there are small errors in the positions. The predictions are sometimes more chaotic, as shown in Fig. 19b, where again some of the deeper atoms are missed and there are some false predictions. The false positives often make it difficult to distinguish the correct predictions. Interestingly, though, the model seems to be predicting the

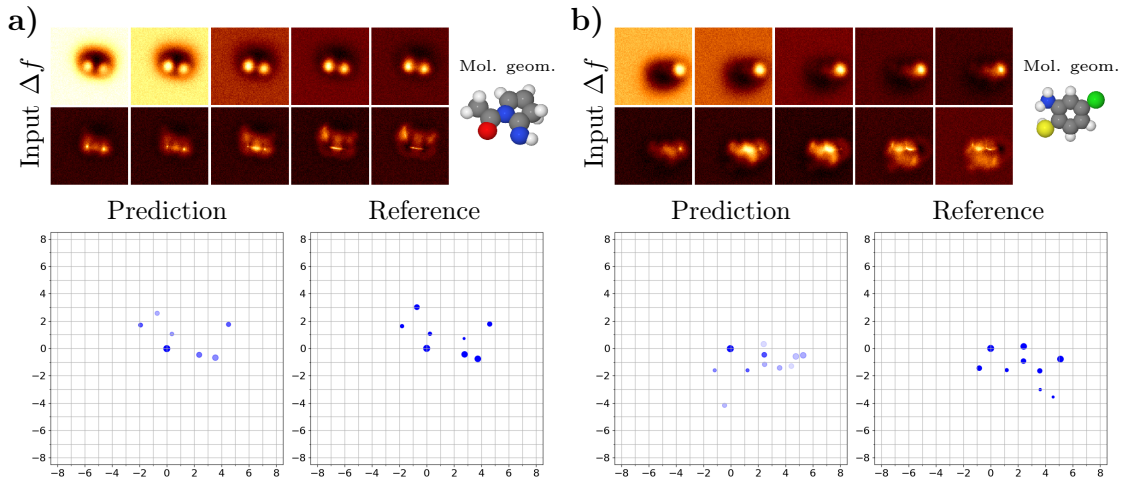

Figure 19: Two example predictions from the xyz model on simulated AFM images. The sizes and transparency of the circles denote the relative depths and confidence levels, respectively. The confidence level threshold for showing the atoms is 0.5.

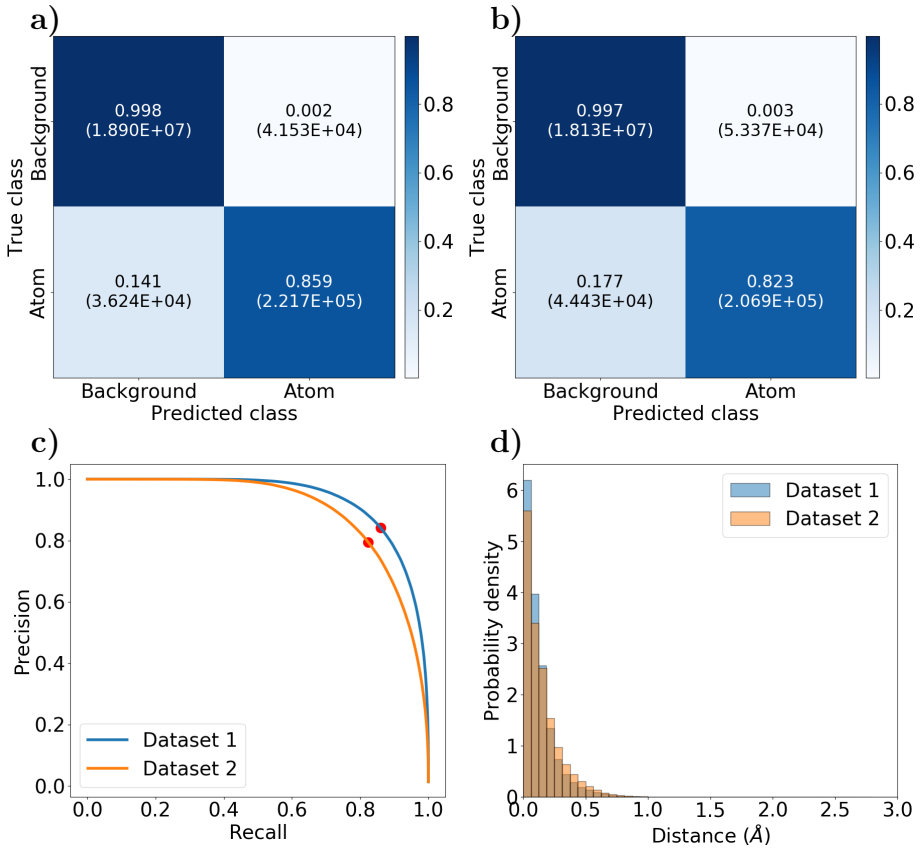

Figure 20: Statistics on the predictions from the xyz model on simulations. **a,b)** Confusion matrices for the classification task in the xyz descriptor at 0.5 confidence level for **a)** dataset 1 and **b)** dataset 2. The values are normalized by dividing the number of instance (in parentheses) by the total number of instances of the corresponding true class. **c)** Precision and recall at different confidence level thresholds. The points denoted by red dots correspond to a confidence level of 0.5. **d)** Histograms of the L^2 error in the position prediction task in the xyz descriptor.

position of the sulfur atom which is not present in the reference for being too deep. This indicates that it could be useful to take into account the effective size of the atoms when making the reference descriptors during training.

The accuracy of the xyz model on the simulations can also be measured quantitatively by statistics on the classification and regression tasks. Figures 20a and 20b show the confusion matrices for the classification task for the two datasets. Most of the predictions go to the *background* class, as would be expected from the class distribution. Most of the *atom*-class boxes are also correctly classified, and the false positives and false negatives are roughly evenly distributed. The rates are slightly better on dataset 1 than dataset 2. This is also seen in the precision-recall curve in Fig. 20c, where the curve for dataset 1 captures slightly more area under the curve. At 0.5 confidence level, the precision and recall for dataset 1 are 0.842 and 0.859, and for dataset 2 are 0.795 and 0.823, respectively. The histograms of the errors in the position prediction in Fig. 20d show that the locations of the atoms are mostly predicted quite accurately. The tail of the distribution is slightly thicker for dataset 2. The mean error and 90th percentile error for dataset 1 are 0.124 Å and 0.278 Å, and for dataset 2 are 0.151 Å and 0.344 Å, respectively. Since the model does not restrict the position prediction by the box size, there are a few outliers that are off by several ångströms.

As with the DSH model, the accuracy of the xyz model on experimental data is hard to gauge reliably. However, we can compare the predictions for the 1S-camphor data to those obtained from the DSH model to see if they are consistent with one another. Figure 21 shows the predictions on the same data as in Fig. 17 at select confidence level thresholds. The highest confidence levels have been selected such that only a few of the highest-confidence atom predictions are shown and the lower-threshold predictions show additionally some of the atoms with lower confidence level. The prediction on experiments 3 looks similar to the prediction from the DSH model with two distinct atoms at the top, except that the atoms are for some reason shifted slightly to the down and left in the xyz prediction. The prediction on experiment 5 is also similar if the down-left atom is neglected in the xyz prediction. The predictions do not seem very informative on the remaining experiments, where only a single atom is predicted at confidence level significantly above the rest, and the atoms of lower confidence level are not very distinctive and do not seem to correspond to anything in the predictions from the DSH model.

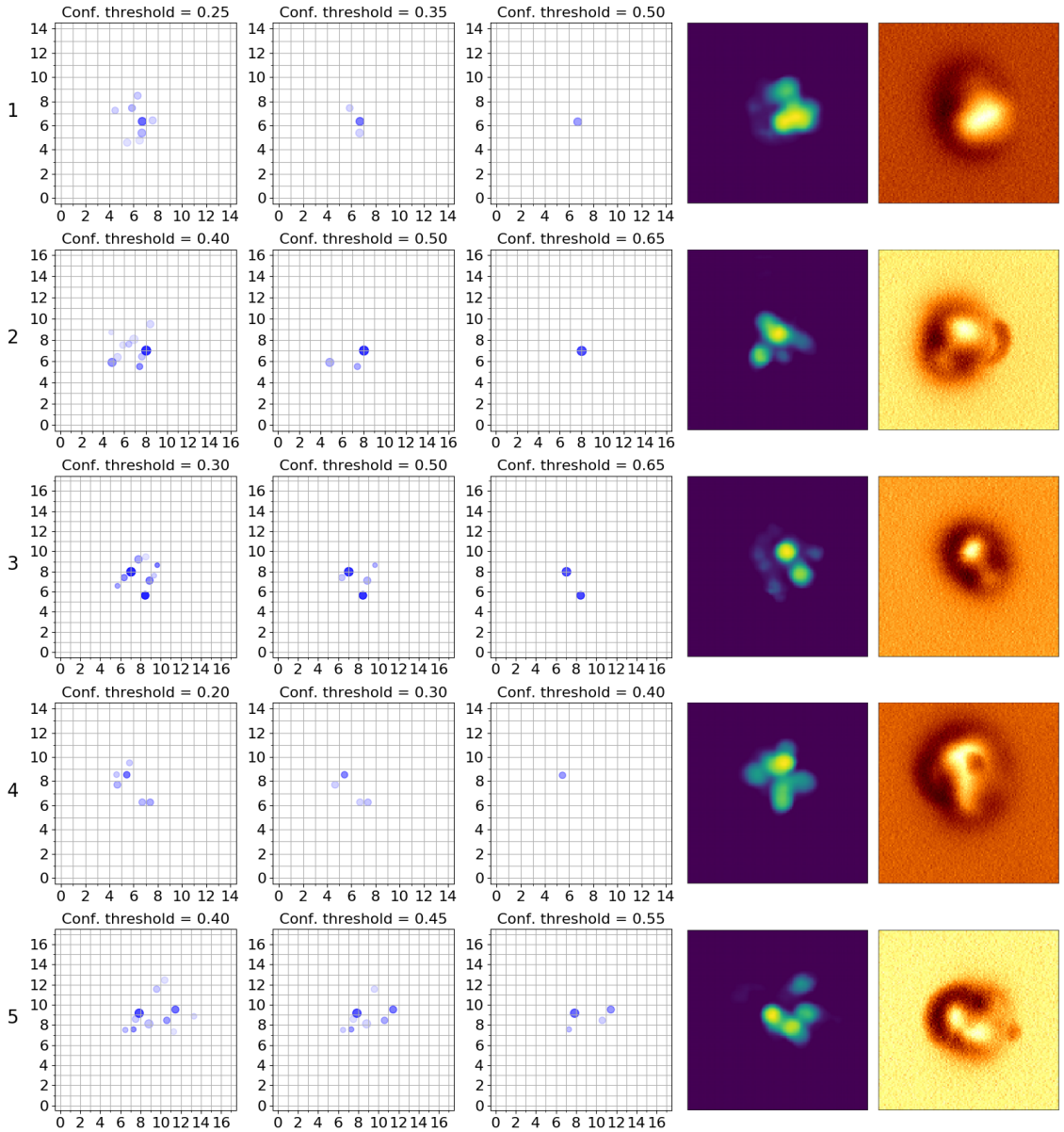

Figure 21: Predictions of the xyz descriptor on experimental data of 1S-camphor. Each row corresponds to one experiment, same as in Fig. 17. Each predictions is shown at three different confidence level thresholds. The sizes and transparency of the circles denote the relative depths and confidence levels, respectively. On the right are columns b and d from Fig. 17 for reference.

5 Discussion and summary

The overall performance of the models on the simulated data was very good. Due to the nature of the AFM imaging process, there was more uncertainty in predictions of the deeper features, but the models in most cases captured the most important features that were to be predicted. Experimental validation was not as straightforward since the correctness of the predictions could not be verified directly. However, the relatively good match between both the predicted descriptors from the DSH model and the corresponding AFM images gives some grounds for confidence in extending the method outside just simulations. The purpose of this was to show an example application for the method and to show that the model can predict something sensible even with experimental data on 3D molecules. Validation on experimental data proved more difficult for the other two models. Firstly, the performance of the ES model could not be experimentally verified due to lack of compatible data. Secondly, despite the relatively good result on simulations, the predictions from the xyz model on experiments did not seem very consistent. Further analysis is required to discover where the discrepancy originates from.

Since the models are trained on simulated data, a natural concern is the accuracy of the simulation. Qualitatively, the Probe Particle Model reproduces well the most distinct features observed in experimental AFM images. However, one important aspect that it does not model, is the movement of the imaged molecule during the scanning process. The molecule could shift laterally on the surface, rotate, or bend slightly when the AFM tip approaches. Likely some of the artefacts we observed in the experimental images here were caused by such mechanical relaxations of the imaged molecule, e.g., panel 5c in Fig. 17. The simulation model also has several parameters that have to be adjusted to match any given experimental setup. We tried to make the deep learning models somewhat resistant to some of these changes by randomly varying some of the scanning parameters when generating and processing the training samples.

Considering the good performance on the simulated data, the models seem quite well suited to the prediction tasks. However there are some aspects that could be improved. The first shortcoming is that we cannot differentiate between atomic species. The Electrostatic Map is already a step towards identifying features beyond just the spatial structure of the molecules. The electrostatic potential can be connected to probable functional groups identifiable by patterns in the potential. One could go further and divide the predicted descriptors based on the elemental number or some other grouping of the atoms. This would fit naturally to the xyz descriptor, where the *atom* class could be divided into further subclasses. This could be aided by using more input channels in the model. We already saw with the ES model that having the same system imaged with two differently functionalized tips helps the accuracy of the prediction. Adding more channels with yet other tip functionalizations or other SPM imaging methods could help in further increasing the accuracy of the predictions.

Secondly, the currently used models are not necessarily very well optimized. There are several hyper-parameters to be optimized in all of the models, including the size and number of filters in each layer, and in the case of the xyz model, the loss

function has hyper-parameters as well. Since the training of a model takes roughly two days, optimizing the hyper-parameters is very challenging.

Thirdly, the DSH and ES models are of the encoder-decoder-CNN type, but they lack one property that the encoder-decoder CNNs typically have, namely skip connections between the encoder and the decoder [14, 15]. Typically the feature map sizes mirror each other in the encoder and decoder, such that the matching levels can be connected together. This allows the passing of gradients to early layers of the network more effectively and helps the decoder in "remembering" the more finely grained features in the original image that could be lost in the low-resolution feature maps of the latent space. We have so far opted not to use skip connections, because the feature maps in the encoder part of our models are 3D, but the feature maps in the decoder are 2D, and the number of filters do not match either. In order to implement the skip connections we would have to transform the 3D feature maps into 2D feature maps. One possibility would be to flatten the 3D feature maps in the z direction with a 1×1 convolution kernel and append the them to the corresponding channel dimension in the decoder layers.

Finally, while our descriptors represent physical properties of the imaged system, the models themselves do not know any physics explicitly. In a way this an advantage of ANNs: they can learn the required knowledge from the data. However, inserting domain knowledge into either the model or the optimization process could prove useful. For example, the vdW Spheres and Atomic Disks descriptors consist of spherical caps and conically decaying disks, but the model is in no way constrained to predict only spherical caps or conically decaying disks, and indeed, it often predicts blurry images where some of the features are fused together. One could imagine modifying the loss function in such a way that the model would be "encouraged" to predict the correct kind of features. In the xyz model we have the additional benefit of actually knowing the exact coordinates of the predicted atoms. These coordinates could be used, for example, to penalize predictions of atoms that are too close on a physical basis. If combined with the knowledge of the atomic elements, more complicated relations between the predicted coordinates could be formed using physical principles.

In summary, we designed CNN models for predicting intuitive descriptors of atomic geometries from AFM images. The models were trained on simulated images and evaluated both on simulated and experimental images. The performance on the simulated images was found to be very good in general. On experimental data, one of the models produced predictions such that the experiments could be matched with simulated images in an automated way. However, on the other models, the predictions either did not seem generally very sensible or experimental validation could not be done. Despite the current challenges, the work in this thesis presents promising first steps towards what could be a powerful technique in investigating molecular systems that were previously out of reach.

References

[1] Niko Pavliček and Leo Gross. "Generation, manipulation and characterization of molecules by atomic force microscopy". In: *Nature Reviews Chemistry* 1 (2017), p. 0005. DOI: 10.1038/s41570-016-0005.

[2] G. Binnig, C. F. Quate, and Ch. Gerber. "Atomic Force Microscope". In: *Phys. Rev. Lett.* 56.9 (1986), pp. 930–933. DOI: 10.1103/PhysRevLett.56.930.

[3] Franz J. Giessibl. "Advances in atomic force microscopy". In: *Rev. Mod. Phys.* 75.3 (2003), pp. 949–983. DOI: 10.1103/RevModPhys.75.949.

[4] Leo Gross, Fabian Mohn, Nikolaj Moll, Peter Liljeroth, and Gerhard Meyer. "The Chemical Structure of a Molecule Resolved by Atomic Force Microscopy". In: *Science* 325.5944 (2009), pp. 1110–1114. DOI: 10.1126/science.1176210.

[5] Leo Gross et al. "Bond-Order Discrimination by Atomic Force Microscopy". In: *Science* 337.6100 (2012), pp. 1326–1329. DOI: 10.1126/science.1225621.

[6] Dimas G. de Oteyza et al. "Direct Imaging of Covalent Bond Structure in Single-Molecule Chemical Reactions". In: *Science* 340.6139 (2013), pp. 1434–1437. DOI: 10.1126/science.1238187.

[7] Leo Gross, Fabian Mohn, Nikolaj Moll, Gerhard Meyer, Rainer Ebel, Wael M. Abdel-Mageed, and Marcel Jaspars. "Organic structure determination using atomic-resolution scanning probe microscopy". In: *Nature Chemistry* 2 (2010), p. 821. DOI: 10.1038/nchem.765.

[8] Florian Albrecht, Niko Pavliček, Coral Herranz-Lancho, Mario Ruben, and Jascha Repp. "Characterization of a Surface Reaction by Means of Atomic Force Microscopy". In: *Journal of the American Chemical Society* 137.23 (2015), pp. 7424–7428. DOI: 10.1021/jacs.5b03114.

[9] Florian Albrecht, Felix Bischoff, Willi Auwärter, Johannes V. Barth, and Jascha Repp. "Direct Identification and Determination of Conformational Response in Adsorbed Individual Nonplanar Molecular Species Using Noncontact Atomic Force Microscopy". In: *Nano Letters* 16.12 (2016), pp. 7703–7709. DOI: 10.1021/acs.nanolett.6b03769.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521 (2015), p. 436. DOI: 10.1038/nature14539.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. 2012, pp. 1097–1105.

[12] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2015).

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[14] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *CoRR* abs/1511.00561 (2015).

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0.

[17] Prokop Hapala, Georgy Kichin, Christian Wagner, F. Stefan Tautz, Ruslan Temirov, and Pavel Jelínek. "Mechanism of high-resolution STM/AFM imaging with functionalized tips". In: *Phys. Rev. B* 90.8 (2014), p. 085421. DOI: 10.1103/PhysRevB.90.085421.

[18] Probe Particle Model code is available at https://github.com/ProkopHapala/ProbeParticleModel.

[19] G. Binnig, H. Rohrer, Ch. Gerber, and E. Weibel. "Surface Studies by Scanning Tunneling Microscopy". In: *Phys. Rev. Lett.* 49.1 (1982), pp. 57–61. DOI: 10.1103/PhysRevLett.49.57.

[20] J. M. Thijssen. *Computational Physics*. New York, NY, USA: Cambridge University Press, 1999. ISBN: 0-521-57588-5.

[21] Adri C. T. van Duin, Siddharth Dasgupta, Francois Lorant, and William A. Goddard. "ReaxFF: A Reactive Force Field for Hydrocarbons". In: *The Journal of Physical Chemistry A* 105.41 (2001), pp. 9396–9409. DOI: 10.1021/jp004368u.

[22] Franz Giessibl. "A direct method to calculate tip-sample forces from frequency shifts in frequency-modulation atomic force microscopy". In: *Applied Physics Letters* 78 (2001), pp. 123–125. DOI: 10.1063/1.1335546.

[23] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016).

[24] Kyunghyun Cho, Bart van Merrienboer, Caglar Gülcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *EMNLP*. 2014.

[25] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2015).

[26] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316 (2016).

[27] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), p. 529. DOI: 10.1038/nature14236.

[28] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *CoRR* abs/1712.01815 (2017).

[29] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[30] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* 65 (1958), pp. 65–386.

[31] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[32] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[33] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[35] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: https://doi.org/10.1016/0893-6080(91)90009-T.

[36] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. “The Expressive Power of Neural Networks: A View from the Width”. In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 6231–6239.

[37] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann Lecun, and Christoph Bregler. “Efficient object localization using Convolutional Networks”. In: 2015, pp. 648–656. DOI: 10.1109/CVPR.2015.7298664.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[39] William L. Jorgensen, Jeffry D. Madura, and Carol J. Swenson. “Optimized intermolecular potential functions for liquid hydrocarbons”. In: *Journal of the American Chemical Society* 106.22 (1984), pp. 6638–6646. DOI: 10.1021/ja00334a030.

[40] Khronos Group. OpenCL programming framework. Website: https://www.khronos.org/opencl/.

[41] Robert M. Parrish et al. “Psi4 1.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability”. In: *Journal of Chemical Theory and Computation* 13.7 (2017), pp. 3185–3197. DOI: 10.1021/acs.jctc.7b00174.

[42] Albert P. Bartók, Risi Kondor, and Gábor Csányi. “On representing chemical environments”. In: *Phys. Rev. B* 87.18 (2013), p. 184115. DOI: 10.1103/PhysRevB.87.184115.

[43] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. “Learning Spatiotemporal Features with 3D Convolutional Networks”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497. DOI: 10.1109/ICCV.2015.510.

[44] D. Maturana and S. Scherer. “VoxNet: A 3D Convolutional Neural Network for real-time object recognition”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928. DOI: 10.1109/IROS.2015.7353481.

[45] François Chollet et al. *Keras*. https://keras.io. 2015.

[46] Martin Abadi et al. “TensorFlow: A system for large-scale machine learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[47] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *CoRR* abs/1603.07285 (2016).

[48] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and Checkerboard Artifacts”. In: *Distill* (2016). DOI: 10.23915/distill.00003.

[49] Prokop Hapala et al. “Mapping the electrostatic force field of single molecules from high-resolution scanning probe images”. In: *Nature Communications* 7 (2016), p. 11560. DOI: 10.1038/ncomms11560.

[50] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. “Focal loss for dense object detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018), pp. 1–1. DOI: 10.1109/TPAMI.2018.2858826.

[51] Roger L. DeKock and Harry B. Gray. *Chemical Structure and Bonding*. University Science Books, 1989. ISBN: 978-0-935702-61-3.

[52] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[53] Terrance Devries and Graham W. Taylor. “Improved Regularization of Convolutional Neural Networks with Cutout”. In: *CoRR* abs/1708.04552 (2017).

[54] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. “Random Erasing Data Augmentation”. In: *CoRR* abs/1708.04896 (2017).

[55] Benjamin Alldritt et al. “Automated Structure Discovery in Atomic Force Microscopy”. In: *ArXiv* abs/1905.10204 (2019).